Nano-Hexapod - Test Bench

Dehaeze Thomas

July 2, 2021

Contents

This document is dedicated to the experimental study of the nano-hexapod shown in Figure 0.1.

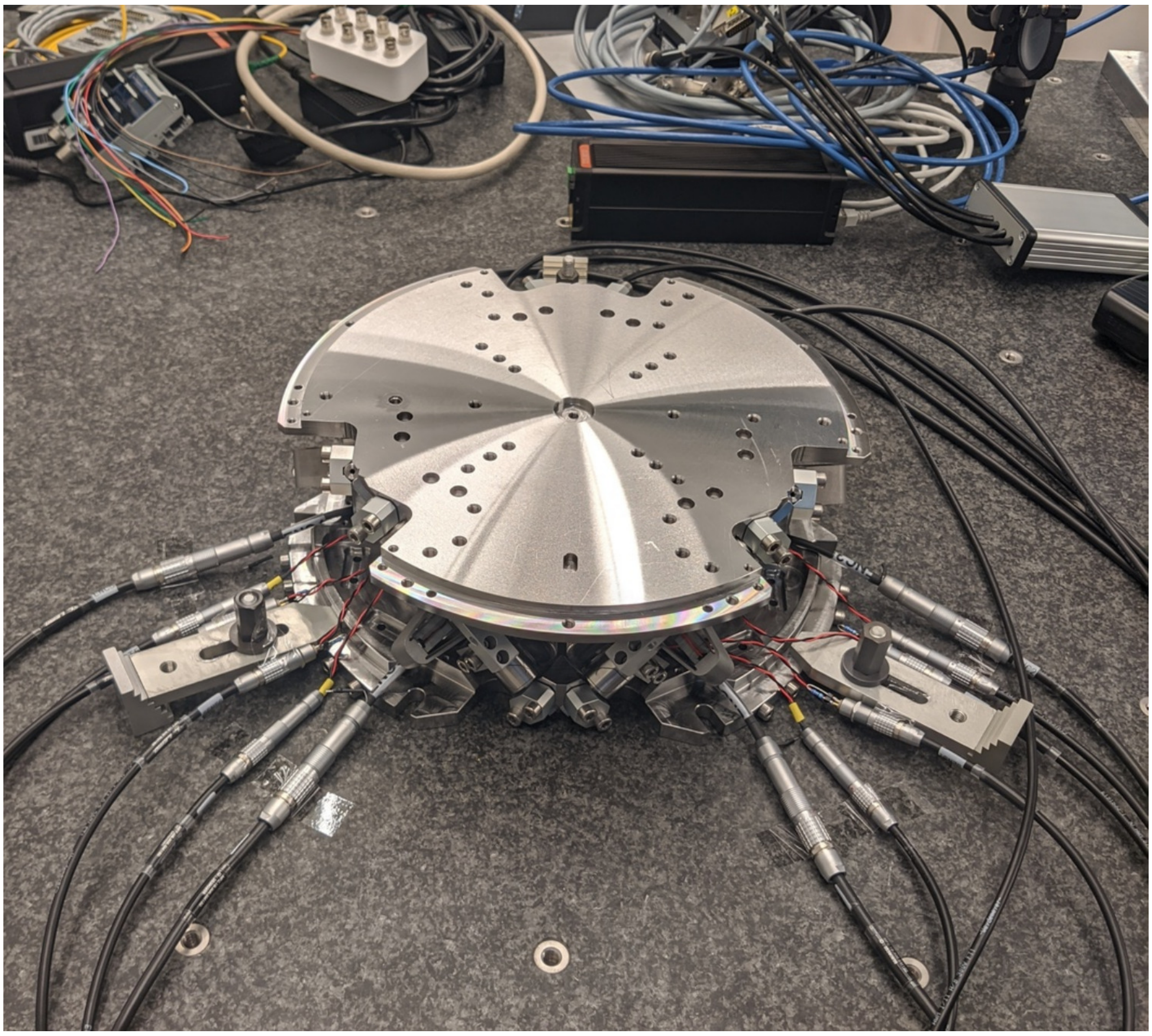


Figure 0.1: Nano-Hexapod

Note

Here are the documentation of the equipment used for this test bench (lots of them are shwon in Figure 0.2):

- Voltage Amplifier: PiezoDrive PD200
- Amplified Piezoelectric Actuator: Cedrat APA300ML
- DAC/ADC: Speedgoat IO313
- Encoder: Renishaw Vionic and used Ruler
- Interferometers: Attocube

In Figure 0.3 is shown a block diagram of the experimental setup. When possible, the notations are consistent with this diagram and summarized in Table 0.1.

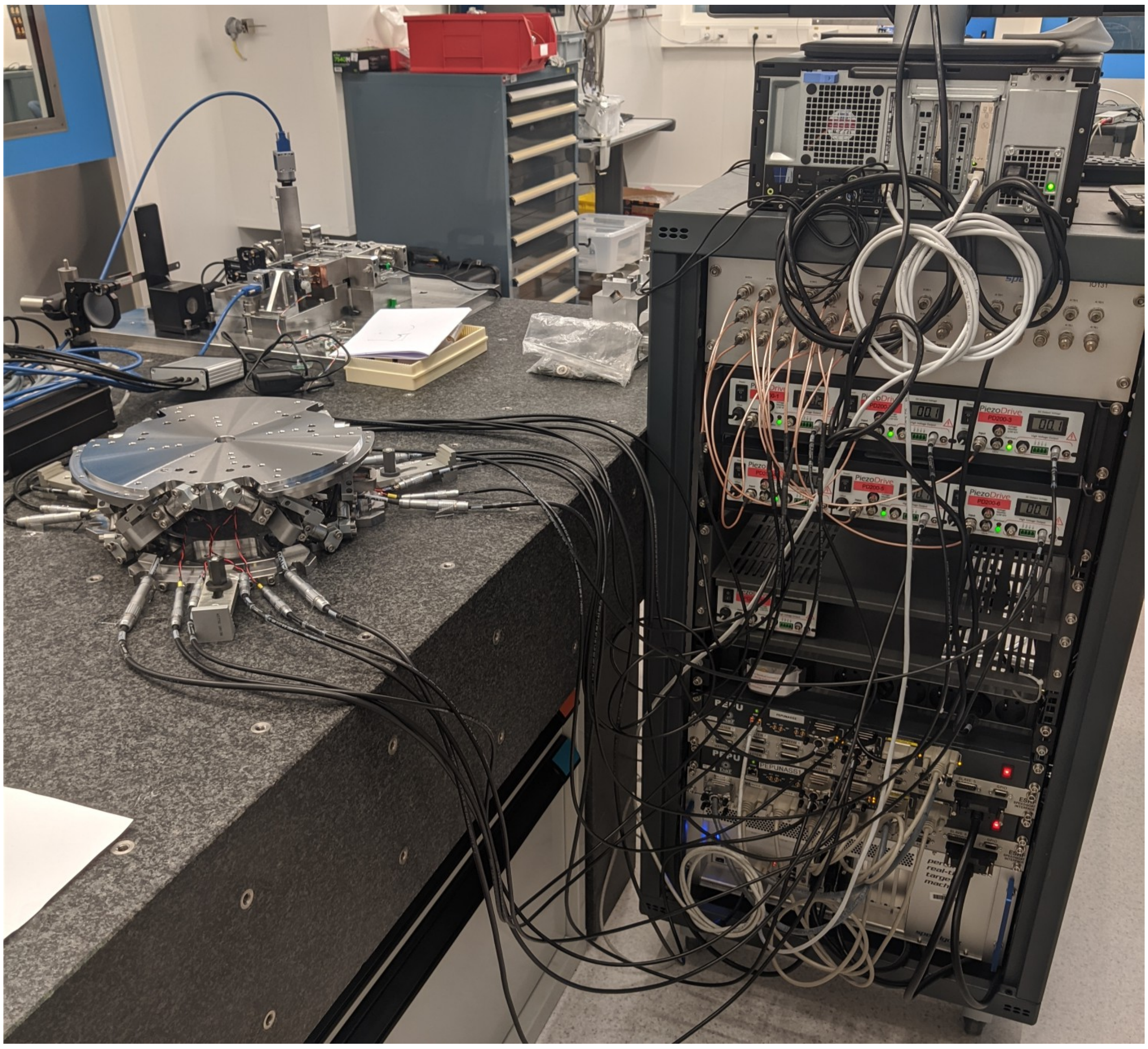


Figure 0.2: Nano-Hexapod and the control electronics

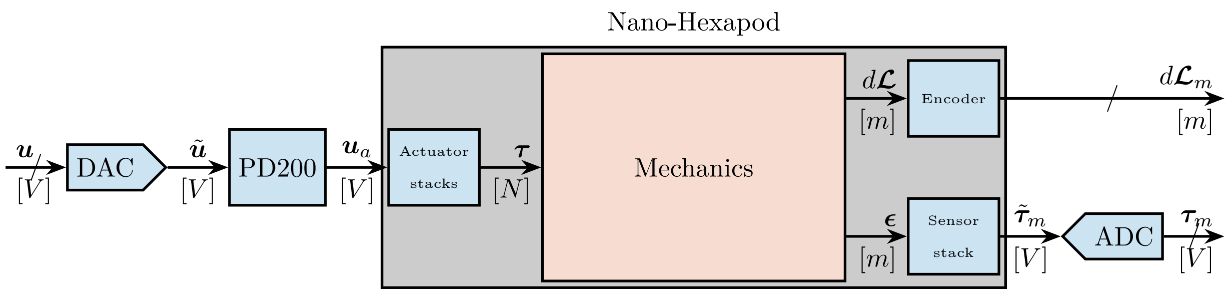


Figure 0.3: Block diagram of the system with named signals

Table 0.1: List of signals

	Unit	**Matlab**	**Vector**	**Elements**
Control Input (wanted DAC voltage)	`[V]`	`u`	$\boldsymbol{u}$	u_i
DAC Output Voltage	`[V]`	`u`	$\tilde{\boldsymbol{u}}$	$\tilde{u}_i$
PD200 Output Voltage	`[V]`	`ua`	$\boldsymbol{u}_a$	$u_{a,i}$
Actuator applied force	`[N]`	`tau`	$\boldsymbol{\tau}$	τ_i
Strut motion	`[m]`	`dL`	$d\boldsymbol{\mathcal{L}}$	$d\mathcal{L}_i$
Encoder measured displacement	`[m]`	`dLm`	$d\boldsymbol{\mathcal{L}}_m$	$d\mathcal{L}_{m,i}$
Force Sensor strain	`[m]`	`epsilon`	$\boldsymbol{\epsilon}$	ϵ_i
Force Sensor Generated Voltage	`[V]`	`taum`	$\tilde{\boldsymbol{\tau}}_m$	$\tilde{\tau}_{m,i}$
Measured Generated Voltage	`[V]`	`taum`	$\boldsymbol{\tau}_m$	$\tau_{m,i}$
Motion of the top platform	`[m,rad]`	`dX`	$d\boldsymbol{\mathcal{X}}$	$d\mathcal{X}_i$
Metrology measured displacement	`[m,rad]`	`dXm`	$d\boldsymbol{\mathcal{X}}_m$	$d\mathcal{X}_{m,i}$

This document is divided in the following sections:

- Section 1: the dynamics of the nano-hexapod when the encoders are fixed to the struts is studied.
- Section 2: the same is done when the encoders are fixed to the plates.
- Section 3: a decentralized HAC-LAC strategy is studied and implemented.

1 Encoders fixed to the Struts - Dynamics

In this section, the encoders are fixed to the struts.

It is divided in the following sections:

- Section 1.1: the transfer function matrix from the actuators to the force sensors and to the encoders is experimentally identified.
- Section 1.2: the obtained FRF matrix is compared with the dynamics of the simscape model
- Section 1.3: decentralized Integral Force Feedback (IFF) is applied and its performances are evaluated.
- Section 1.4: a modal analysis of the nano-hexapod is performed

1.1 Identification of the dynamics

1.1.1 Load Measurement Data

Matlab
```matlab
%% Load Identification Data
meas_data_lf = {};

for i = 1:6
    meas_data_lf(i) = {load(sprintf('mat/frf_data_exc_strut_%i_noise_lf.mat', i), 't', 'Va', 'Vs', 'de')};
    meas_data_hf(i) = {load(sprintf('mat/frf_data_exc_strut_%i_noise_hf.mat', i), 't', 'Va', 'Vs', 'de')};
end
```

1.1.2 Spectral Analysis - Setup

Matlab
```matlab
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_data_lf{1}.t(end) - (meas_data_lf{1}.t(1)))/(length(meas_data_lf{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_data_lf{1}.Va, meas_data_lf{1}.de, win, [], [], 1/Ts);
```

```
i_lf = f < 250; % Points for low frequency excitation
i_hf = f > 250; % Points for high frequency excitation
```

1.1.3 Transfer function from Actuator to Encoder

First, let's compute the coherence from the excitation voltage and the displacement as measured by the encoders (Figure 1.1).

```matlab
%% Coherence
coh_dvf = zeros(length(f), 6, 6);

for i = 1:6
    coh_dvf_lf = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
    coh_dvf_hf = mscohere(meas_data_hf{i}.Va, meas_data_hf{i}.de, win, [], [], 1/Ts);
    coh_dvf(:,:,i) = [coh_dvf_lf(i_lf, :); coh_dvf_hf(i_hf, :)];
end
```

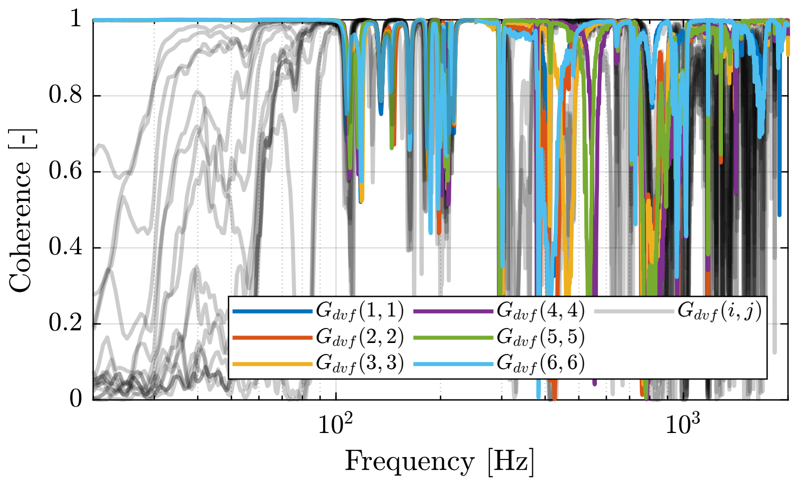


Figure 1.1: Obtained coherence for the DVF plant

Then the 6x6 transfer function matrix is estimated (Figure 1.2).

```matlab
%% DVF Plant (transfer function from u to dLm)
G_dvf = zeros(length(f), 6, 6);

for i = 1:6
    G_dvf_lf = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
    G_dvf_hf = tfestimate(meas_data_hf{i}.Va, meas_data_hf{i}.de, win, [], [], 1/Ts);
    G_dvf(:,:,i) = [G_dvf_lf(i_lf, :); G_dvf_hf(i_hf, :)];
end
```

1.1.4 Transfer function from Actuator to Force Sensor

First, let's compute the coherence from the excitation voltage and the displacement as measured by the encoders (Figure 1.3).

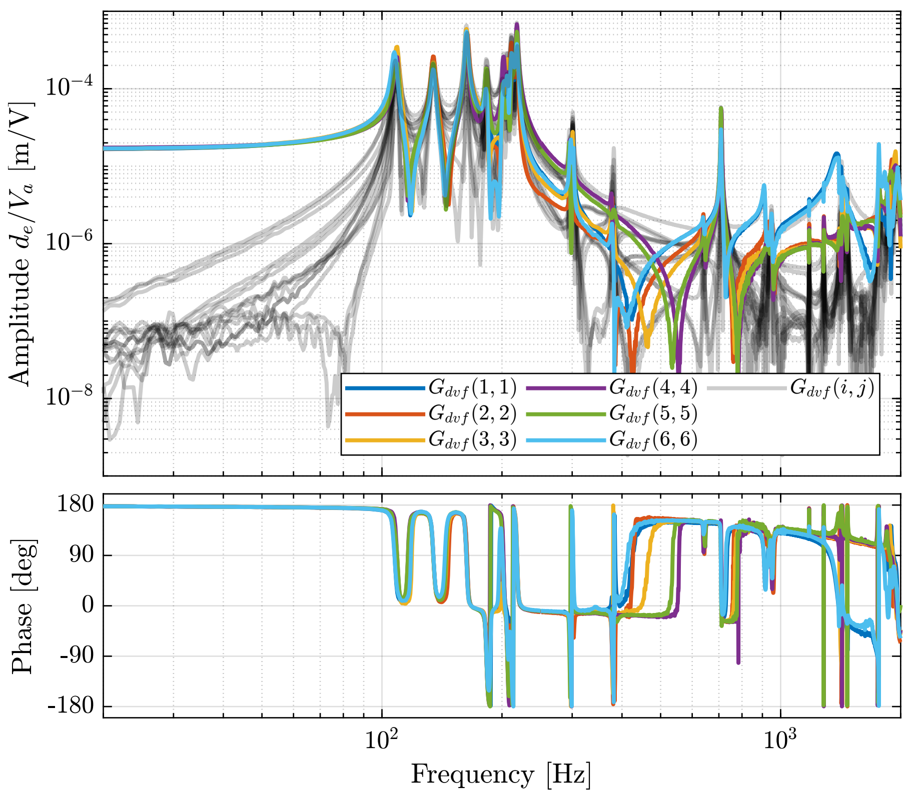


Figure 1.2: Measured FRF for the DVF plant

Matlab
```
%% Coherence for the IFF plant
coh_iff = zeros(length(f), 6, 6);

for i = 1:6
    coh_iff_lf = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
    coh_iff_hf = mscohere(meas_data_hf{i}.Va, meas_data_hf{i}.Vs, win, [], [], 1/Ts);
    coh_iff(:,:,i) = [coh_iff_lf(i_lf, :); coh_iff_hf(i_hf, :)];
end
```

Then the 6x6 transfer function matrix is estimated (Figure 1.4).

Matlab
```
%% IFF Plant
G_iff = zeros(length(f), 6, 6);

for i = 1:6
    G_iff_lf = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
    G_iff_hf = tfestimate(meas_data_hf{i}.Va, meas_data_hf{i}.Vs, win, [], [], 1/Ts);
    G_iff(:,:,i) = [G_iff_lf(i_lf, :); G_iff_hf(i_hf, :)];
end
```

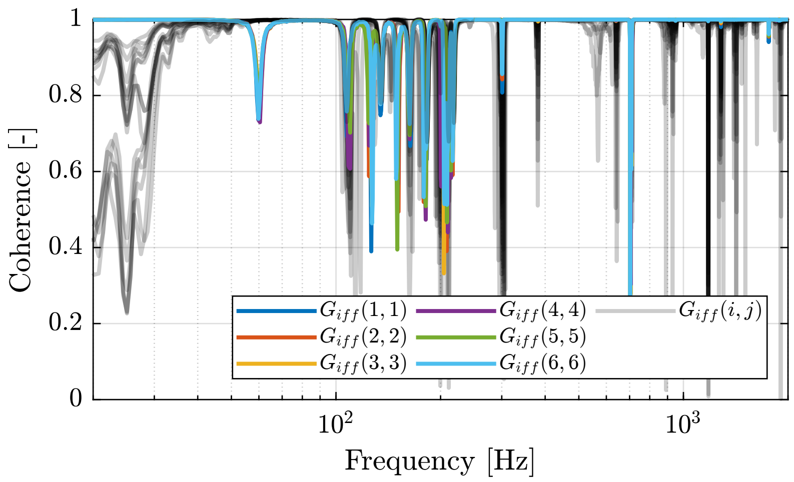


Figure 1.3: Obtained coherence for the IFF plant

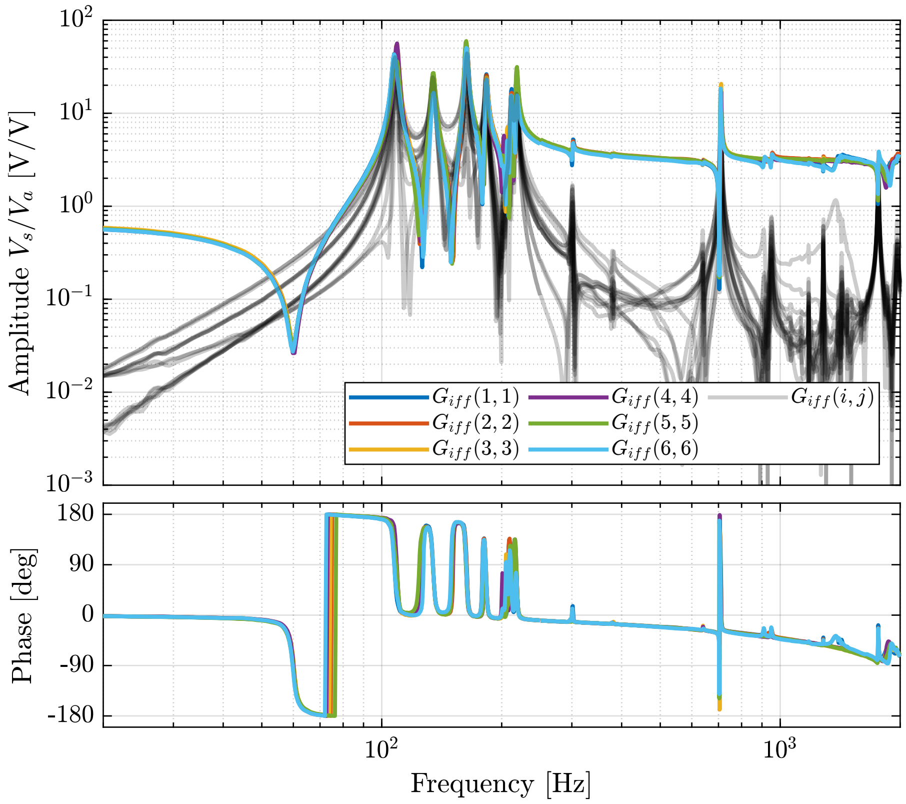


Figure 1.4: Measured FRF for the IFF plant

1.1.5 Save Identified Plants

```matlab
save('matlab/mat/identified_plants_enc_struts.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

1.2 Comparison with the Simscape Model

In this section, the measured dynamics is compared with the dynamics estimated from the Simscape model.

1.2.1 Load measured FRF

```matlab
%% Load data
load('identified_plants_enc_struts.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

1.2.2 Dynamics from Actuator to Force Sensors

```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', '2dof');
```

```matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

1.2.3 Dynamics from Actuator to Encoder

```matlab
%% Initialization of the Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible');
```

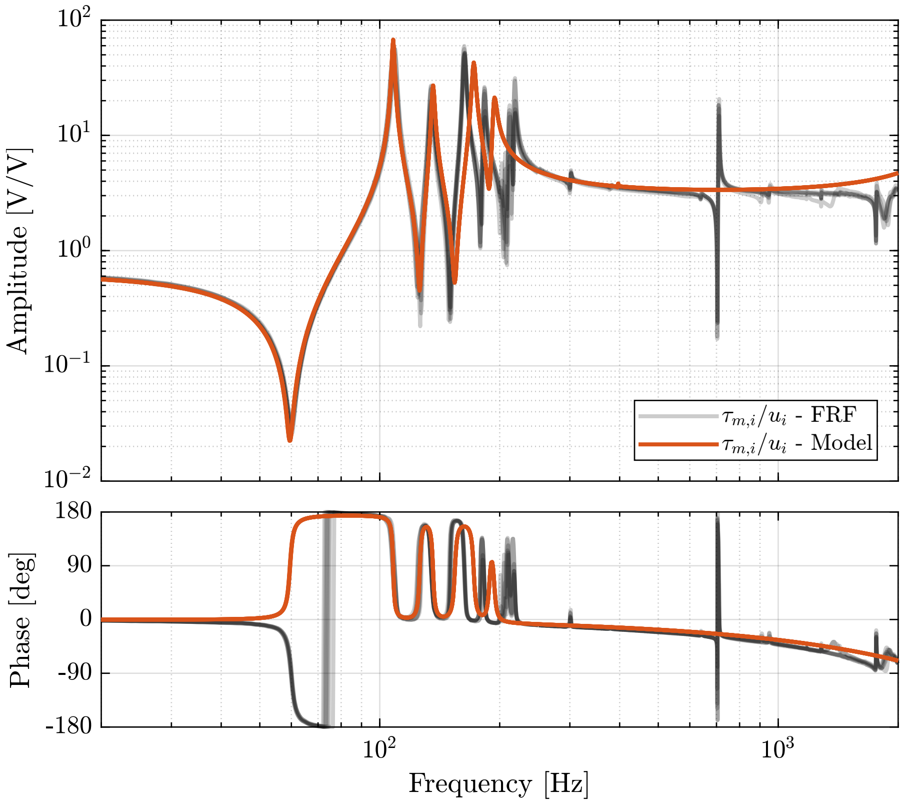


Figure 1.5: Diagonal elements of the IFF Plant

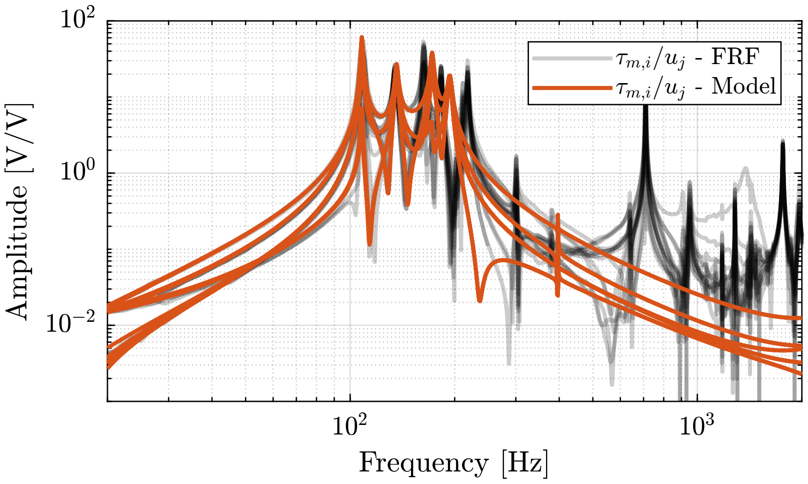


Figure 1.6: Off diagonal elements of the IFF Plant

```
Matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders

Gdvf = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

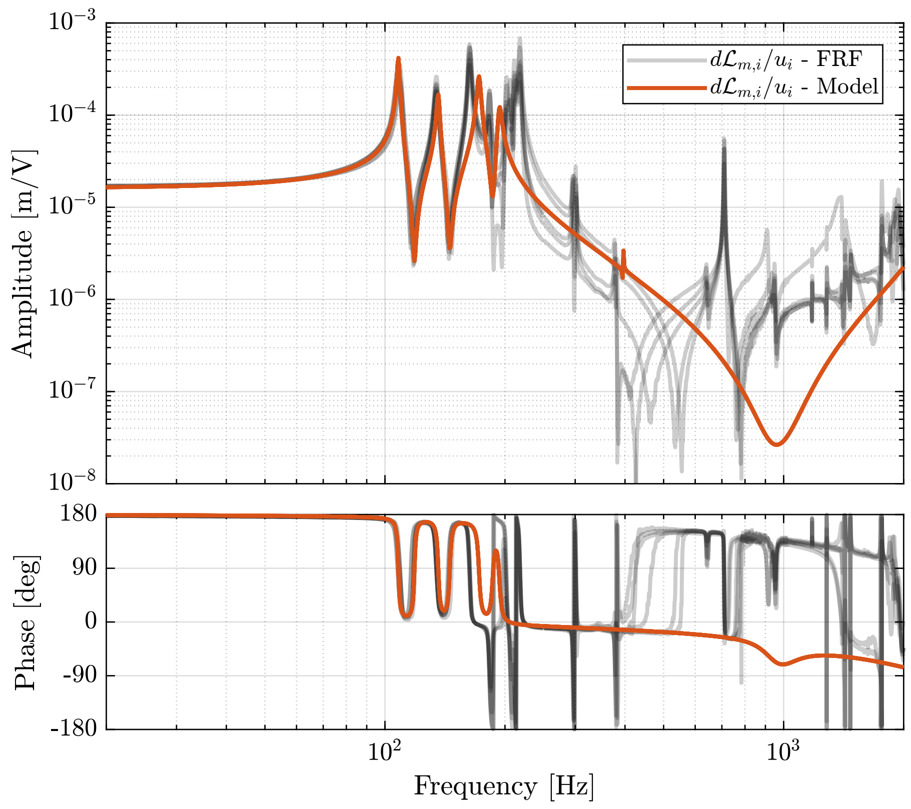


Figure 1.7: Diagonal elements of the DVF Plant

1.2.4 Effect of a change in bending damping of the joints

```
Matlab
%% Tested bending dampings [Nm/(rad/s)]
cRs = [1e-3, 5e-3, 1e-2, 5e-2, 1e-1];
```

```
Matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

Then the identification is performed for all the values of the bending damping.

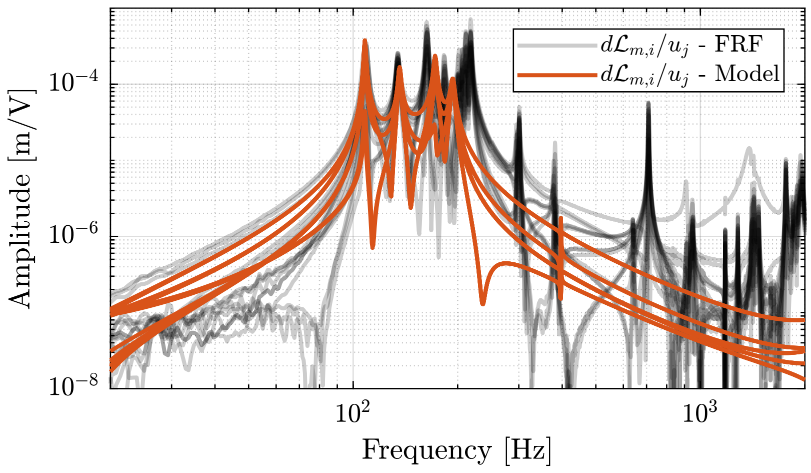


Figure 1.8: Off diagonal elements of the DVF Plant

```matlab
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(cRs), 1)};

for i = 1:length(cRs)
    n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                           'flex_top_type', '4dof', ...
                                           'motion_sensor_type', 'struts', ...
                                           'actuator_type', 'flexible', ...
                                           'flex_bot_cRx', cRs(i), ...
                                           'flex_bot_cRy', cRs(i), ...
                                           'flex_top_cRx', cRs(i), ...
                                           'flex_top_cRy', cRs(i));

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

- Could be nice
- Actual damping is very small

1.2.5 Effect of a change in damping factor of the APA

```matlab
%% Tested bending dampings [Nm/(rad/s)]
xis = [1e-3, 5e-3, 1e-2, 5e-2, 1e-1];
```

```matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

```matlab
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(xis), 1)};

for i = 1:length(xis)
    n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                           'flex_top_type', '4dof', ...
                                           'motion_sensor_type', 'struts', ...
                                           'actuator_type', 'flexible', ...
                                           'actuator_xi', xis(i));

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

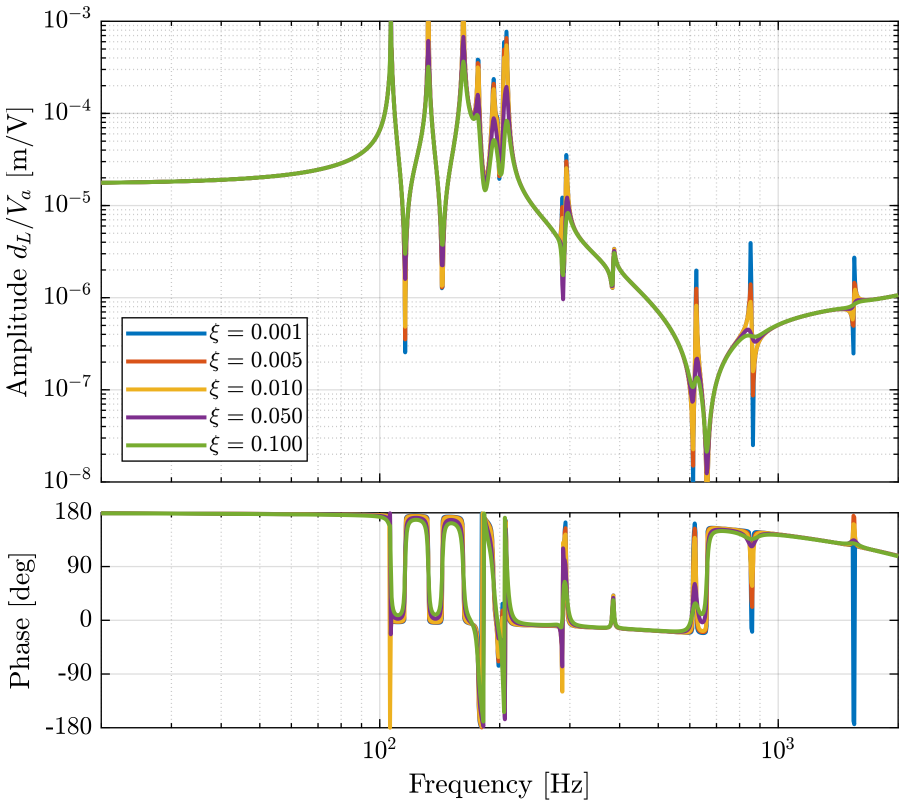


Figure 1.9: Effect of the APA damping factor ξ on the dynamics from u to $d\mathcal{L}$

Important

Damping factor ξ has a large impact on the damping of the “spurious resonances” at 200Hz and 300Hz.

Question

Why is the damping factor does not change the damping of the first peak?

1.2.6 Effect of a change in stiffness damping coef of the APA

Matlab
```matlab
m_coef = 1e1;
```

Matlab
```matlab
%% Tested bending dampings [Nm/(rad/s)]
k_coefs = [1e-6, 5e-6, 1e-5, 5e-5, 1e-4];
```

Matlab
```matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

Matlab
```matlab
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(k_coefs), 1)};
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible');

for i = 1:length(k_coefs)
    k_coef = k_coefs(i);

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

1.2.7 Effect of a change in mass damping coef of the APA

Matlab
```matlab
k_coef = 1e-6;
```

Matlab
```matlab
%% Tested bending dampings [Nm/(rad/s)]
m_coefs = [1e1, 5e1, 1e2, 5e2, 1e3];
```

Matlab
```matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

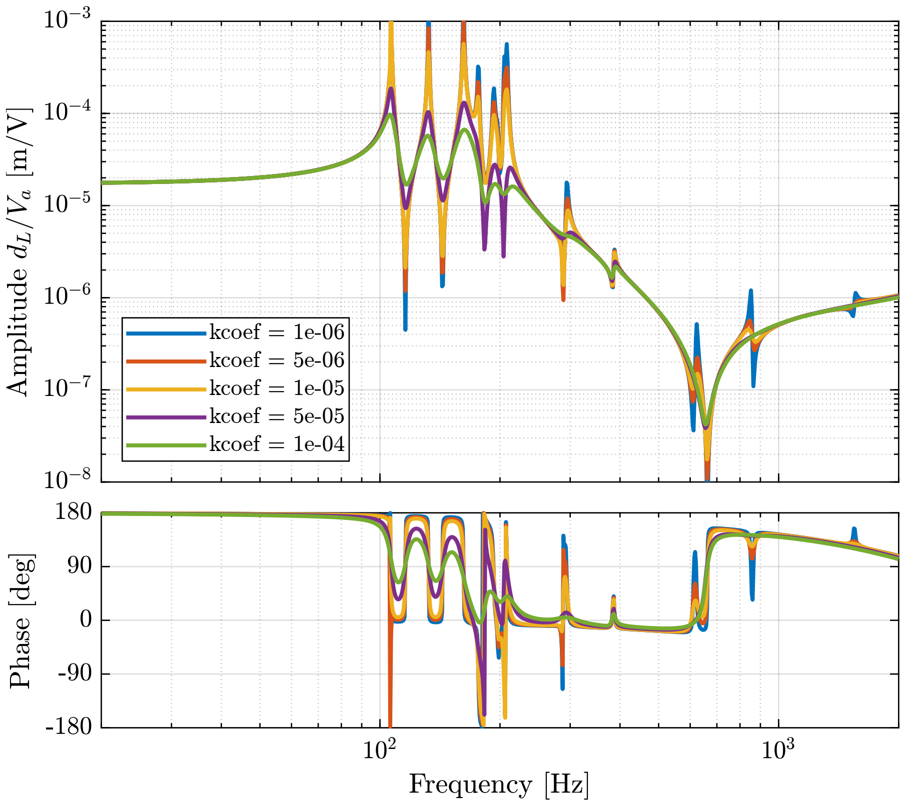


Figure 1.10: Effect of a change of the damping "stiffness coeficient" on the transfer function from u to $d\mathcal{L}$

```
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(m_coefs), 1)};
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible');

for i = 1:length(m_coefs)
    m_coef = m_coefs(i);

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

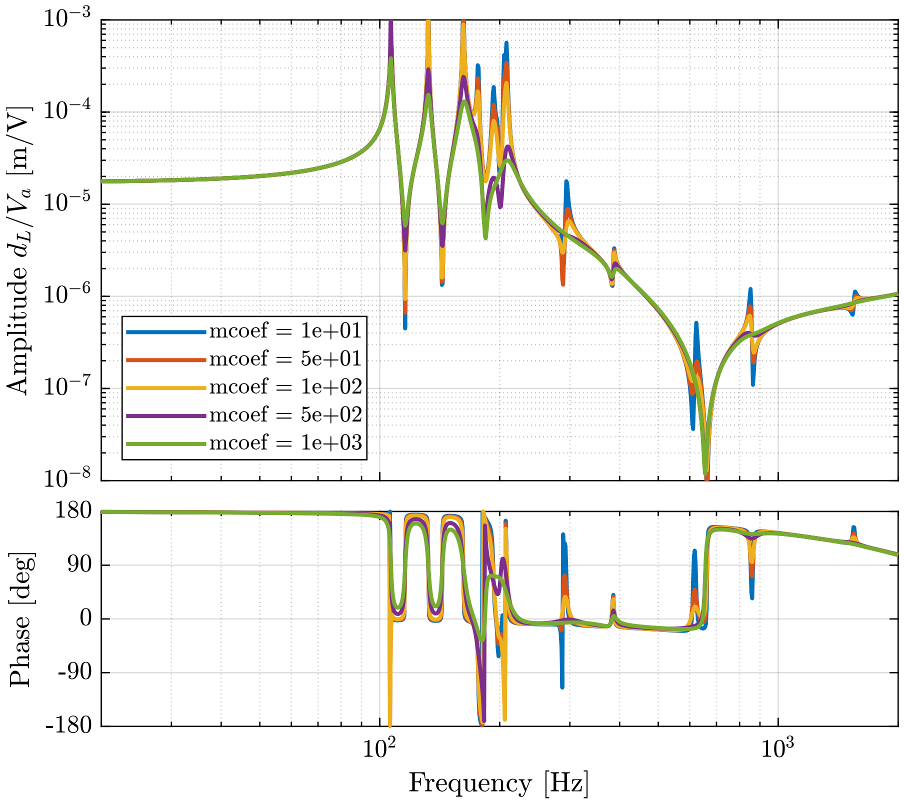


Figure 1.11: Effect of a change of the damping "mass coeficient" on the transfer function from u to $d\mathcal{L}$

1.2.8 Using Flexible model

```
d_aligns = [[-0.05,  -0.3,  0];
            [ 0,      0.5,  0];
            [-0.1,   -0.3,  0];
            [ 0,      0.3,  0];
            [-0.05,   0.05, 0];
            [0,       0,    0]]*1e-3;
```

```matlab
d_aligns = zeros(6,3);
% d_aligns(1,:) = [-0.05,  -0.3,  0]*1e-3;
d_aligns(2,:) = [ 0,      0.3,  0]*1e-3;
```

```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns);
```

Question

Why do we have smaller resonances when using flexible APA? On the test bench we have the same resonance as the 2DoF model. Could it be due to the compliance in other dof of the flexible model?

```matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders

Gdvf = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

```matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

1.2.9 Flexible model + encoders fixed to the plates

```matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors
```

```matlab
d_aligns = [[-0.05,  -0.3,  0];
            [ 0,      0.5,  0];
            [-0.1,   -0.3,  0];
            [ 0,      0.3,  0];
            [-0.05,   0.05, 0];
            [0,       0,    0]]*1e-3;
```

```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns);
```

```matlab
Gdvf_struts = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns);
```

```matlab
Gdvf_plates = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

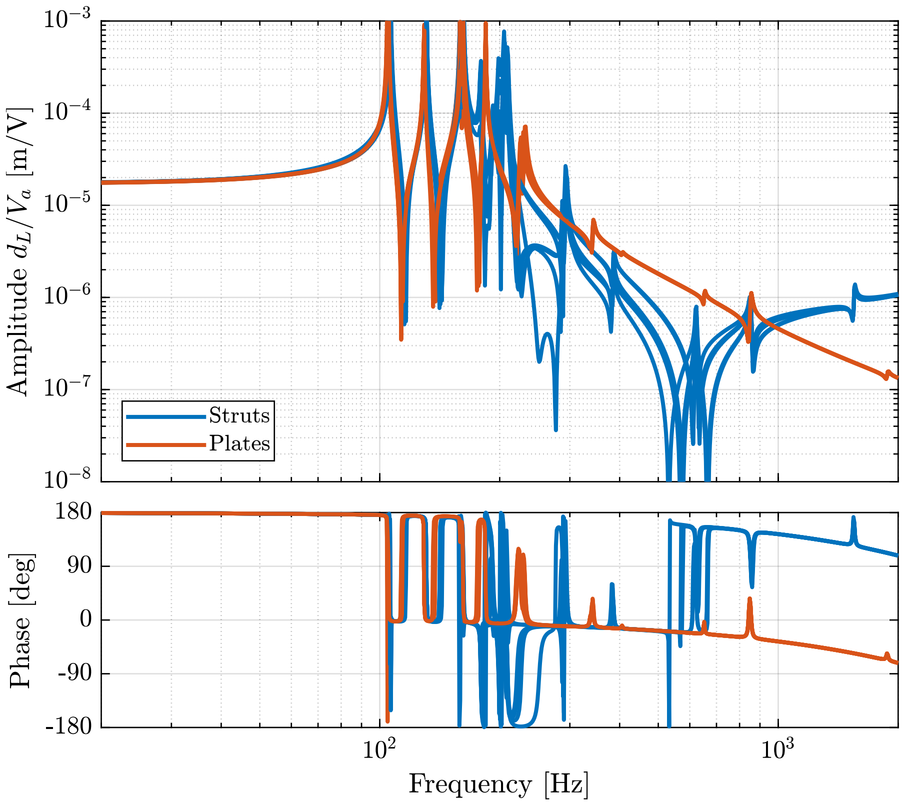


Figure 1.12: Comparison of the dynamics from V_a to d_L when the encoders are fixed to the struts (blue) and to the plates (red). APA are modeled as a flexible element.

1.3 Integral Force Feedback

In this section, the Integral Force Feedback (IFF) control strategy is applied to the nano-hexapod. The main goal of this to add damping to the nano-hexapod's modes.

The control architecture is shown in Figure 1.13 where $\boldsymbol{K}_{\text{IFF}}$ is a diagonal 6×6 controller.

The system as then a new input $\boldsymbol{u}'$, and the transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ should be easier to control than the initial transfer function from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$.

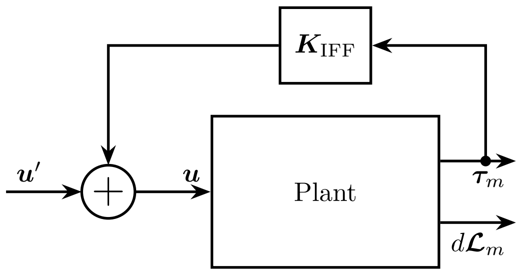


Figure 1.13: Integral Force Feedback Strategy

This section is structured as follow:

- Section 1.3.1: Using the Simscape model (APA taken as 2DoF model), the transfer function from $\boldsymbol{u}$ to $\boldsymbol{\tau}_m$ is identified. Based on the obtained dynamics, the control law is developed and the optimal gain is estimated using the Root Locus.
- Section 1.3.2: Still using the Simscape model, the effect of the IFF gain on the the transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ is studied.
- Section 1.3.3: The same is performed experimentally: several IFF gains are used and the damped plant is identified each time.
- Section 1.3.4: The damped model and the identified damped system are compared for the optimal IFF gain. It is found that IFF indeed adds a lot of damping into the system. However it is not efficient in damping the spurious struts modes.
- Section 1.3.5: Finally, a "flexible" model of the APA is used in the Simscape model and the optimally damped model is compared with the measurements.

1.3.1 IFF Control Law and Optimal Gain

Let's use a model of the Nano-Hexapod with the encoders fixed to the struts and the APA taken as 2DoF model.

Matlab
```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', '2dof');
```

The transfer function from $\boldsymbol{u}$ to $\boldsymbol{\tau}_m$ is identified.

```
Matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

The IFF controller is defined as shown below:

```
Matlab
%% IFF Controller
Kiff_g1 = -(1/(s + 2*pi*40))*...      % LPF: provides integral action above 40Hz
           (s/(s + 2*pi*30))*...      % HPF: limit low frequency gain
           (1/(1 + s/2/pi/500))*...   % LPF: more robust to high frequency resonances
           eye(6);                    % Diagonal 6x6 controller
```

Then, the poles of the system are shown in the complex plane as a function of the controller gain (i.e. Root Locus plot) in Figure 1.14. A gain of 400 is chosen as the "optimal" gain as it visually seems to be the gain that adds the maximum damping to all the suspension modes simultaneously.

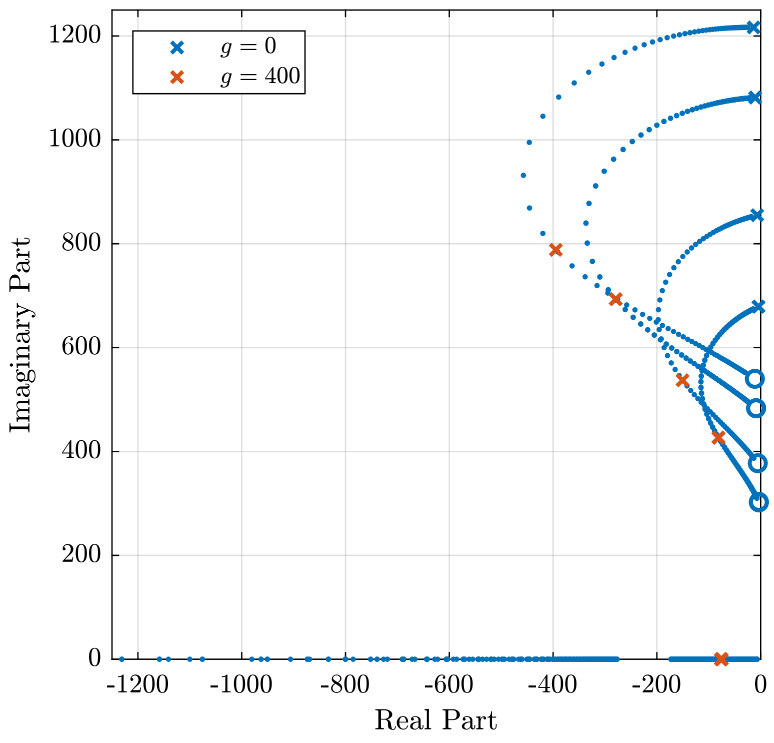


Figure 1.14: Root Locus for the IFF control strategy

Then the "optimal" IFF controller is:

```
Matlab
%% IFF controller with Optimal gain
Kiff = 400*Kiff_g1;
```

And it is saved for further use.

```
save('mat/Kiff.mat', 'Kiff')
```

The bode plots of the "diagonal" elements of the loop gain are shown in Figure 1.15. It is shown that the phase and gain margins are quite high and the loop gain is large arround the resonances.

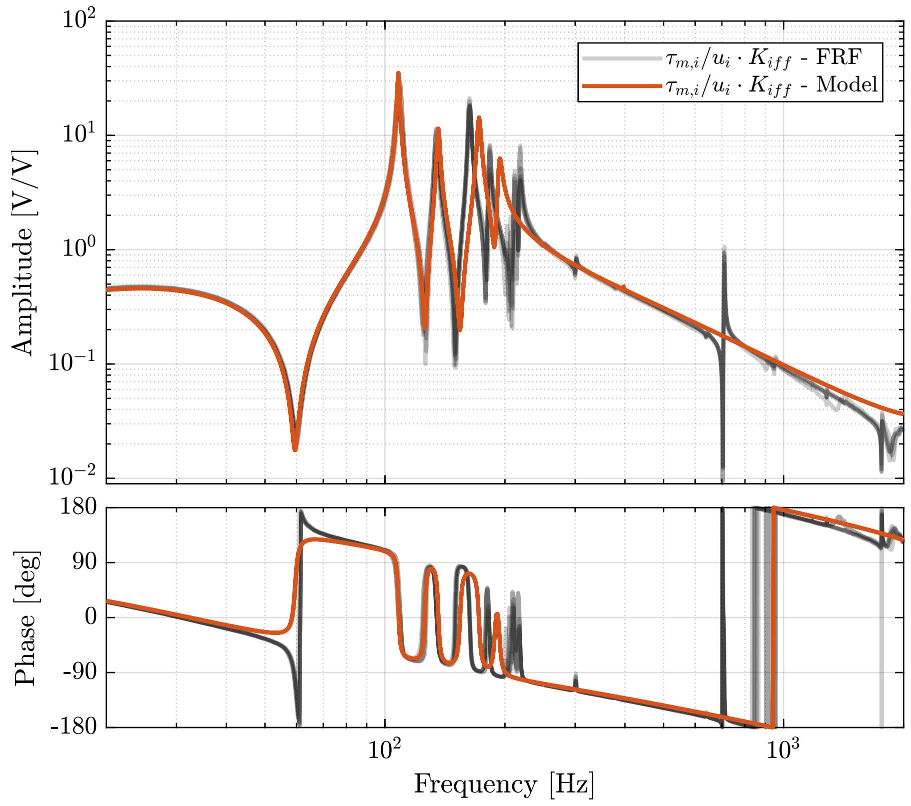


Figure 1.15: Bode plot of the "decentralized loop gain" $G_{\text{iff}}(i,i) \times K_{\text{iff}}(i,i)$

1.3.2 Effect of IFF on the plant - Simulations

Still using the Simscape model with encoders fixed to the struts and 2DoF APA, the IFF strategy is tested.

```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', '2dof', ...
                                       'controller_type', 'iff');
```

The following IFF gains are tried:

```matlab
%% Tested IFF gains
iff_gains = [4, 10, 20, 40, 100, 200, 400];
```

And the transfer functions from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ are identified for all the IFF gains.

```matlab
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
Gd_iff = {zeros(1, length(iff_gains))};

clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'], 1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Strut Displacement (encoder)

for i = 1:length(iff_gains)
    Kiff = iff_gains(i)*Kiff_g1*eye(6); % IFF Controller
    Gd_iff(i) = {exp(-s*Ts)*linearize(mdl, io, 0.0, options)};

    isstable(Gd_iff{i})
end
```

The obtained dynamics are shown in Figure 1.16.

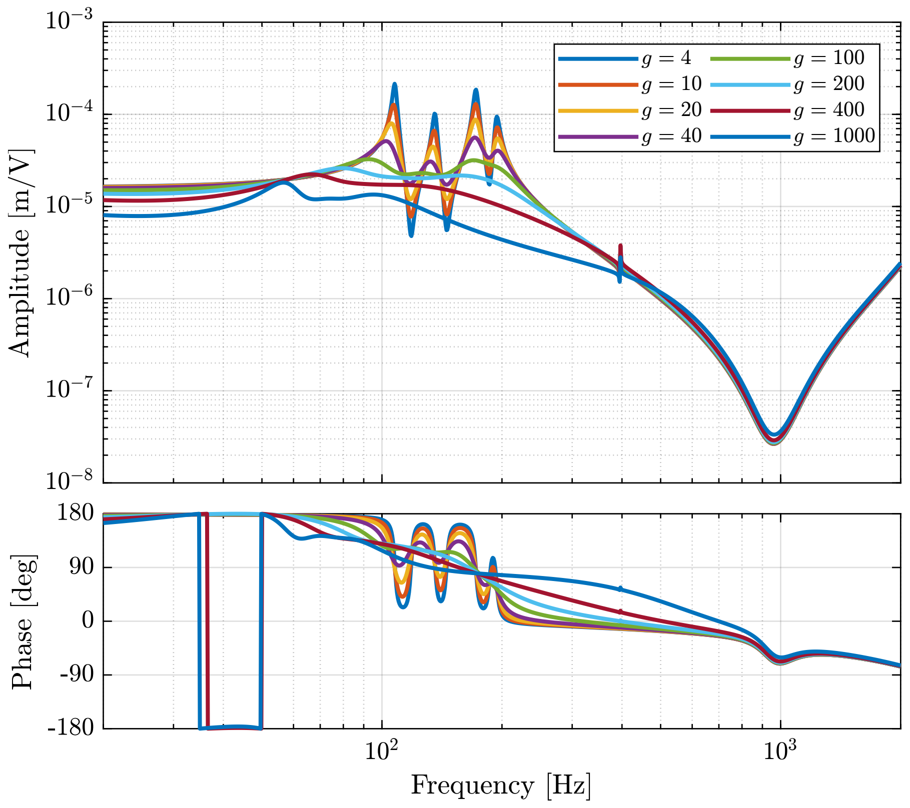


Figure 1.16: Effect of the IFF gain g on the transfer function from $\boldsymbol{\tau}$ to $d\boldsymbol{\mathcal{L}}_m$

1.3.3 Effect of IFF on the plant - Experimental Results

The IFF strategy is applied experimentally and the transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ is identified for all the defined values of the gain.

Load Data

First load the identification data.

Matlab
```
%% Load Identification Data
meas_iff_gains = {};

for i = 1:length(iff_gains)
    meas_iff_gains(i) = {load(sprintf('mat/iff_strut_1_noise_g_%i.mat', iff_gains(i)), 't', 'Vexc', 'Vs', 'de', 'u')};
end
```

Spectral Analysis - Setup

And define the useful variables that will be used for the identification using the `tfestimate` function.

Matlab
```
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_iff_gains{1}.t(end) - (meas_iff_gains{1}.t(1)))/(length(meas_iff_gains{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_iff_gains{1}.Vexc, meas_iff_gains{1}.de, win, [], [], 1/Ts);
```

DVF Plant

The transfer functions are estimated for all the values of the gain.

Matlab
```
%% DVF Plant (transfer function from u to dLm)
G_iff_gains = {};

for i = 1:length(iff_gains)
    G_iff_gains{i} = tfestimate(meas_iff_gains{i}.Vexc, meas_iff_gains{i}.de(:,1), win, [], [], 1/Ts);
end
```

The obtained dynamics as shown in the bode plot in Figure 1.17. The dashed curves are the results obtained using the model, and the solid curves the results from the experimental identification.

The bode plot is then zoomed on the suspension modes of the nano-hexapod in Figure 1.18.

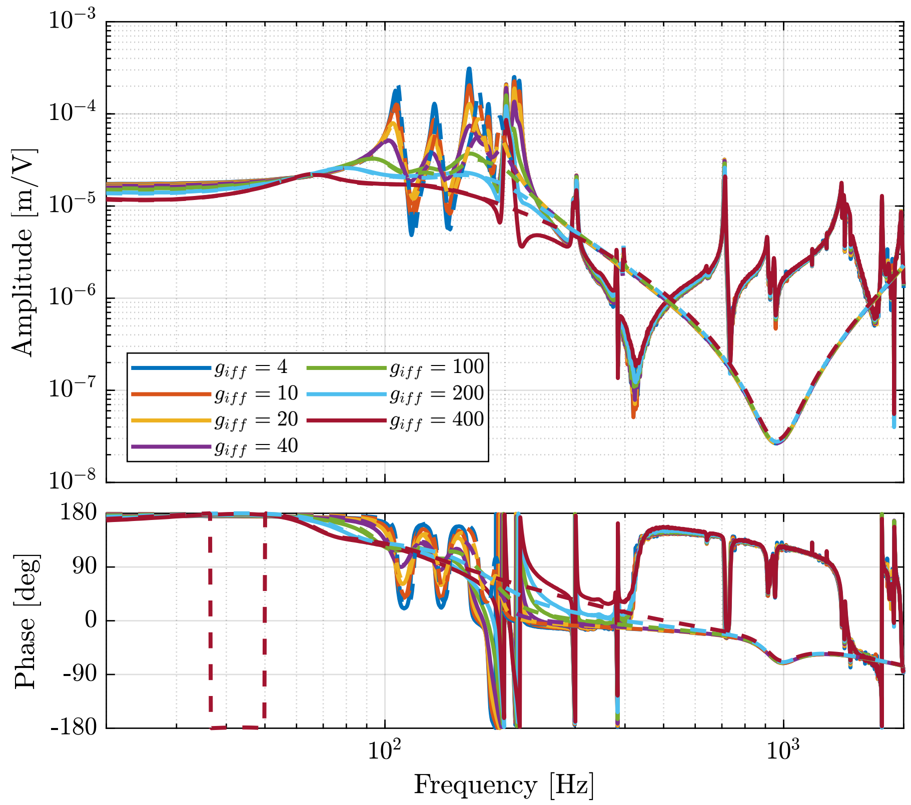


Figure 1.17: Transfer function from u to $d\mathcal{L}_m$ for multiple values of the IFF gain

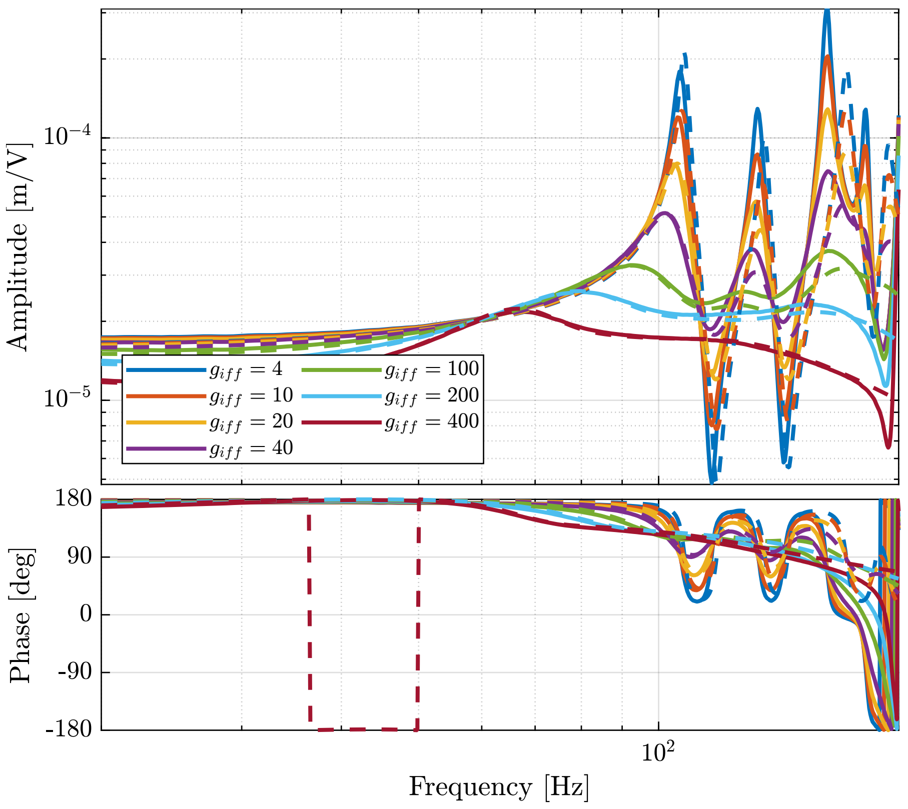


Figure 1.18: Transfer function from u to $d\mathcal{L}_m$ for multiple values of the IFF gain (Zoom)

Important

The IFF control strategy is very effective for the damping of the suspension modes. It however does not damp the modes at 200Hz, 300Hz and 400Hz (flexible modes of the APA).
Also, the experimental results and the models obtained from the Simscape model are in agreement concerning the damped system (up to the flexible modes).

Experimental Results - Comparison of the un-damped and fully damped system

The un-damped and damped experimental plants are compared in Figure 1.19 (diagonal terms).

It is very clear that all the suspension modes are very well damped thanks to IFF. However, there is little to no effect on the flexible modes of the struts and of the plate.

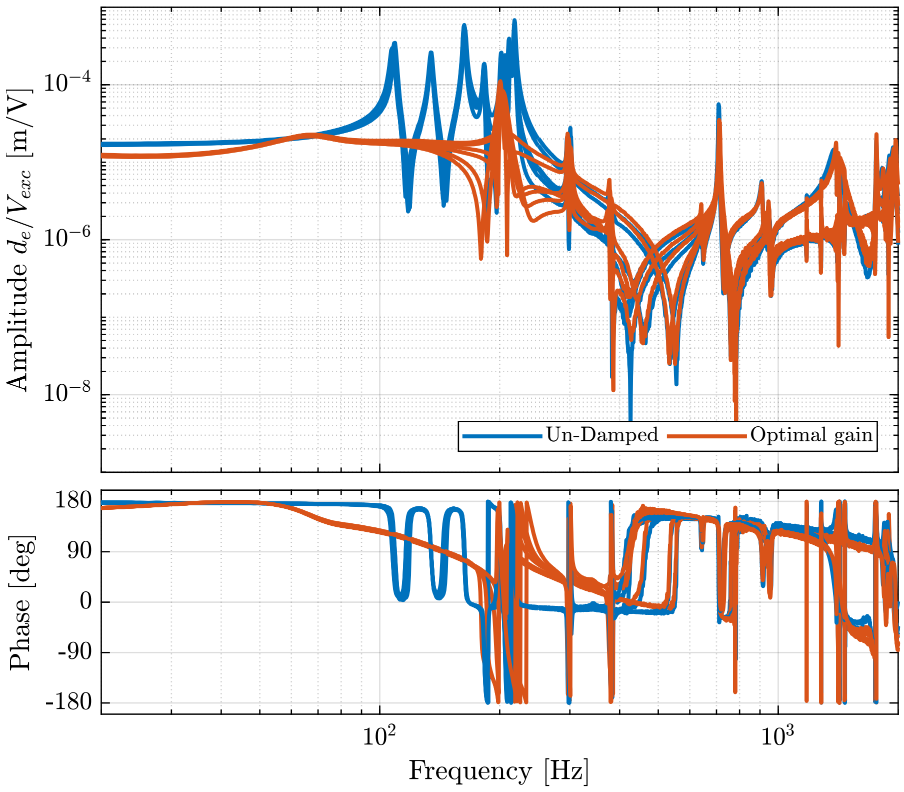


Figure 1.19: Comparison of the diagonal elements of the tranfer function from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ without active damping and with optimal IFF gain

1.3.4 Experimental Results - Damped Plant with Optimal gain

Let's now look at the 6×6 damped plant with the optimal gain $g = 400$.

Load Data

The experimental data are loaded.

```matlab
%% Load Identification Data
meas_iff_struts = {};

for i = 1:6
    meas_iff_struts(i) = {load(sprintf('mat/iff_strut_%i_noise_g_400.mat', i), 't', 'Vexc', 'Vs', 'de', 'u')};
end
```

Spectral Analysis - Setup

And the parameters useful for the spectral analysis are defined.

```matlab
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_iff_struts{1}.t(end) - (meas_iff_struts{1}.t(1)))/(length(meas_iff_struts{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_iff_struts{1}.Vexc, meas_iff_struts{1}.de, win, [], [], 1/Ts);
```

DVF Plant

Finally, the 6×6 plant is identified using the `tfestimate` function.

```matlab
%% DVF Plant (transfer function from u to dLm)
G_iff_opt = {};

for i = 1:6
    G_iff_opt{i} = tfestimate(meas_iff_struts{i}.Vexc, meas_iff_struts{i}.de, win, [], [], 1/Ts);
end
```

The obtained diagonal elements are compared with the model in Figure 1.20.

And all the off-diagonal elements are compared with the model in Figure 1.21.

Important

With the IFF control strategy applied and the optimal gain used, the suspension modes are very well damped. Remains the un-damped flexible modes of the APA (200Hz, 300Hz, 400Hz), and the modes of the plates (700Hz).
The Simscape model and the experimental results are in very good agreement.

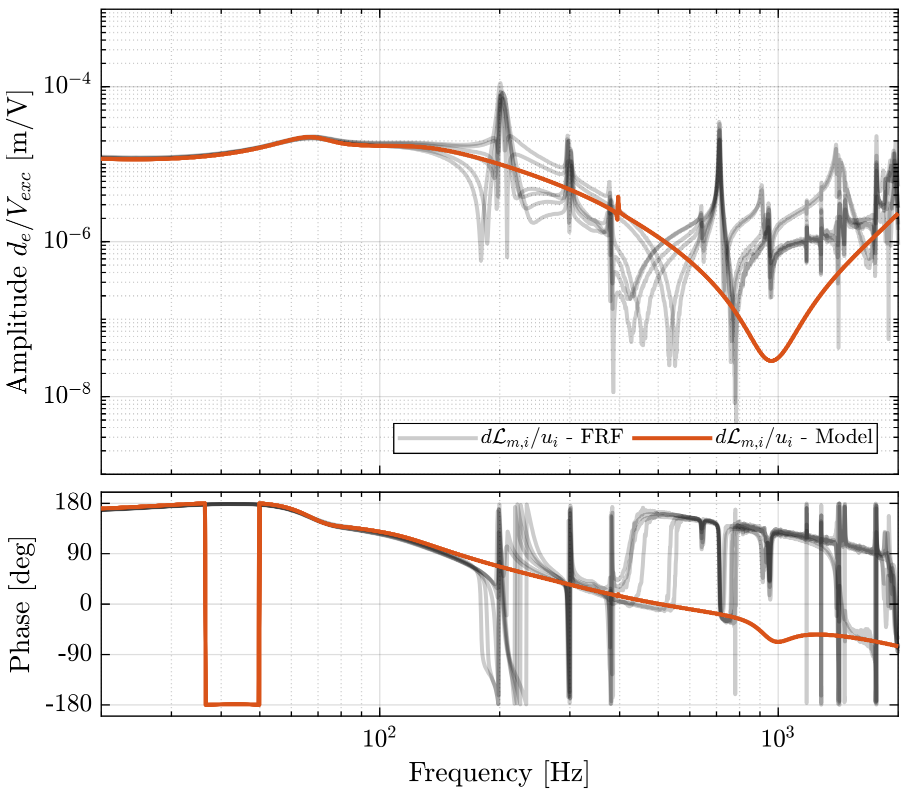


Figure 1.20: Comparison of the diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

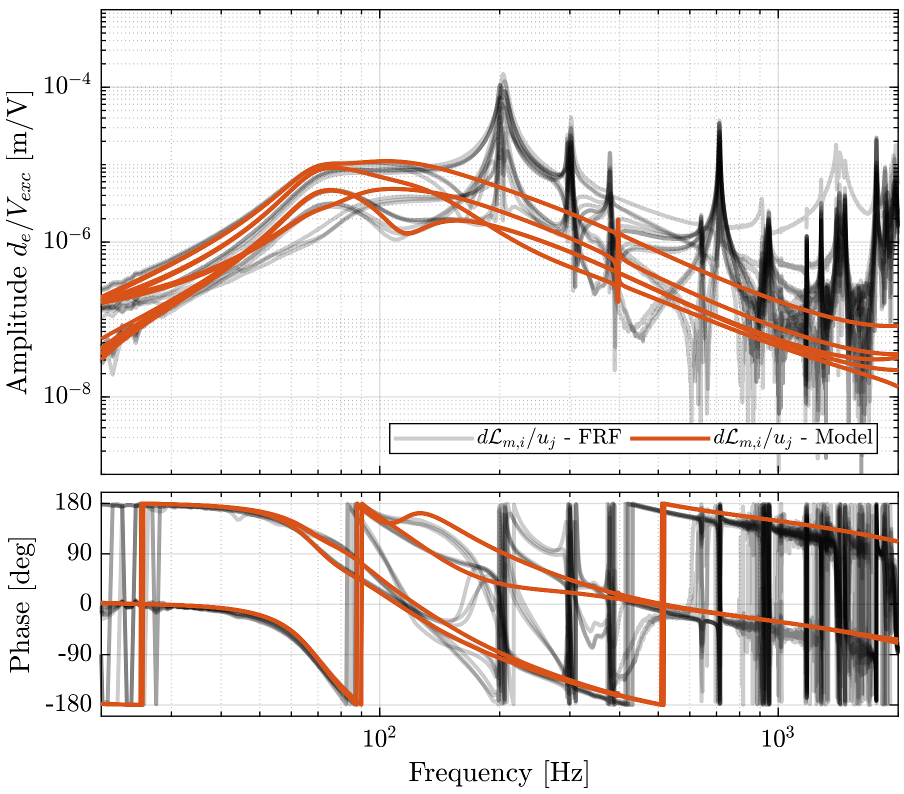


Figure 1.21: Comparison of the off-diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

1.3.5 Comparison with the Flexible model

When using the 2-DoF model for the APA, the flexible modes of the struts were not modelled, and it was the main limitation of the model. Now, let's use a flexible model for the APA, and see if the obtained damped plant using the model is similar to the measured dynamics.

First, the nano-hexapod is initialized.

Matlab
```
%% Estimated misalignement of the struts
d_aligns = [[-0.05,  -0.3,  0];
            [ 0,      0.5,  0];
            [-0.1,   -0.3,  0];
            [ 0,      0.3,  0];
            [-0.05,   0.05, 0];
            [0,       0,    0]]*1e-3;


%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns, ...
                                       'controller_type', 'iff');
```

And the "optimal" controller is loaded.

Matlab
```
%% Optimal IFF controller
load('Kiff.mat', 'Kiff');
```

The transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ is identified using the Simscape model.

Matlab
```
%% Linearized inputs/outputs
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'],  1, 'openoutput'); io_i = io_i + 1; % Strut Displacement (encoder)

%% Identification of the plant
Gd_iff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

The obtained diagonal elements are shown in Figure 1.22 while the off-diagonal elements are shown in Figure 1.23.

Important

Using flexible models for the APA, the agreement between the Simscape model of the nano-hexapod and the measured FRF is very good.
Only the flexible mode of the top-plate is not appearing in the model which is very logical as the top plate is taken as a solid body.

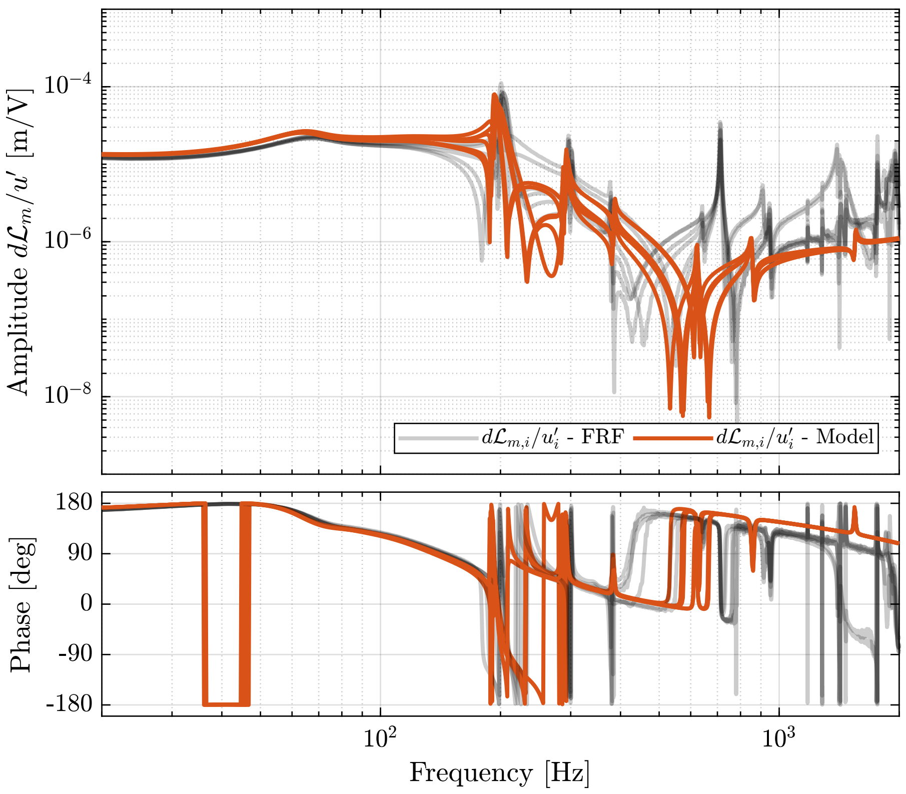


Figure 1.22: Diagonal elements of the transfer function from $\boldsymbol{u}'$ to $d\mathcal{L}_m$ - comparison of the measured FRF and the identified dynamics using the flexible model

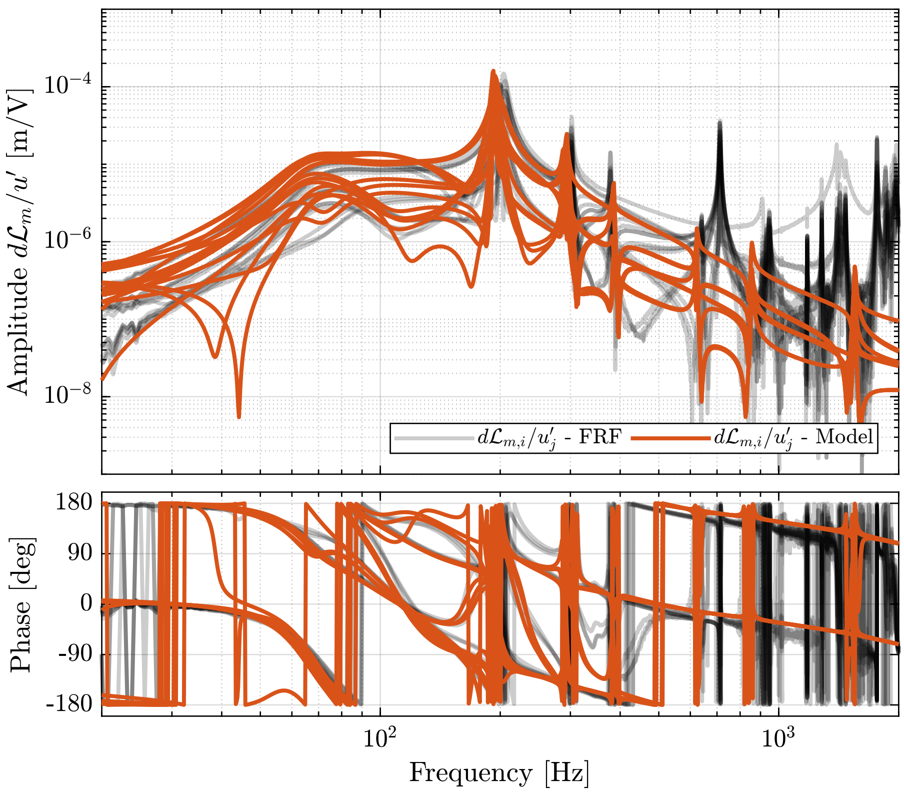


Figure 1.23: Off-diagonal elements of the transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ - comparison of the measured FRF and the identified dynamics using the flexible model

1.3.6 Conclusion

Important

The decentralized Integral Force Feedback strategy applied on the nano-hexapod is very effective in damping all the suspension modes.
The Simscape model (especially when using a flexible model for the APA) is shown to be very accurate, even when IFF is applied.

1.4 Modal Analysis

Several 3-axis accelerometers are fixed on the top platform of the nano-hexapod as shown in Figure 1.28.

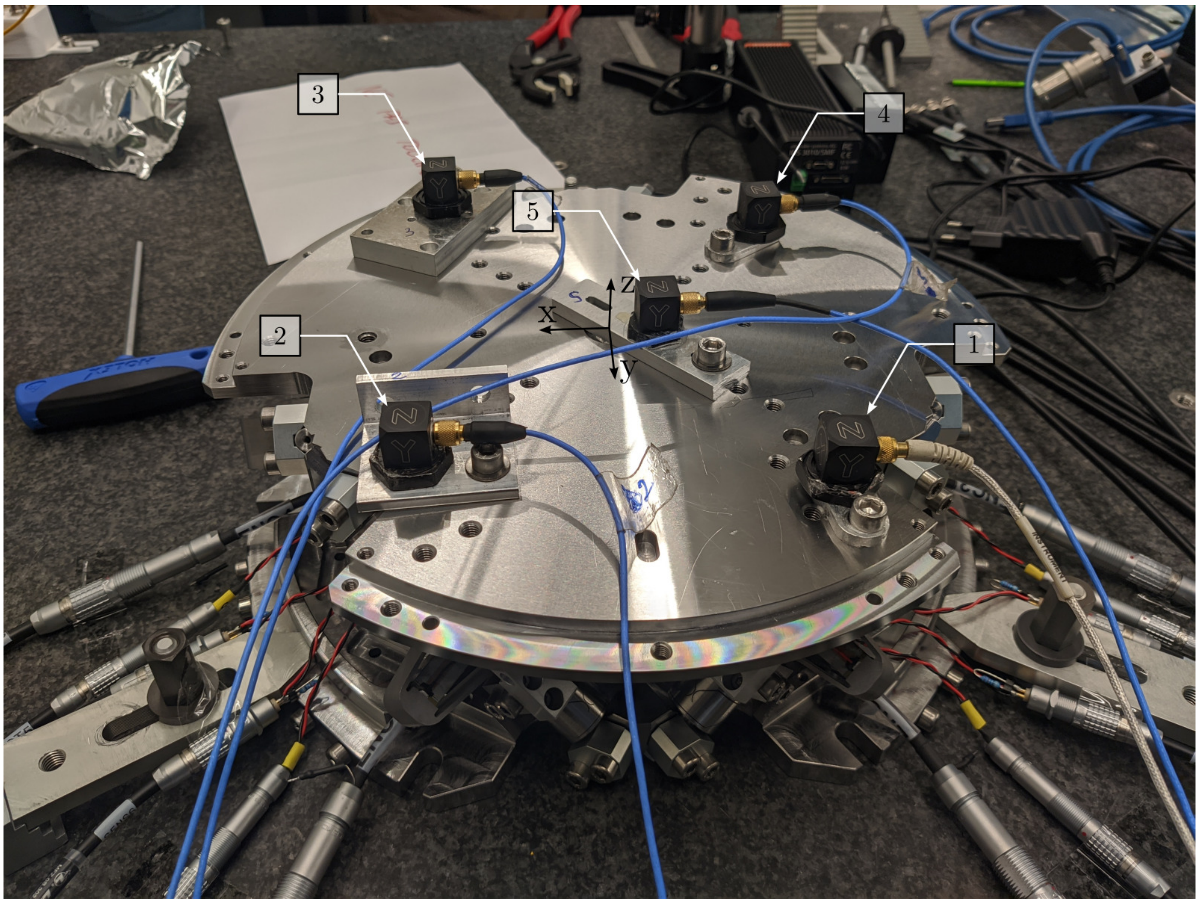


Figure 1.24: Location of the accelerometers on top of the nano-hexapod

The top platform is then excited using an instrumented hammer as shown in Figure 1.25.

From this experiment, the resonance frequencies and the associated mode shapes can be computed (Section 1.4.1). Then, in Section 1.4.2, the vertical compliance of the nano-hexapod is experimentally

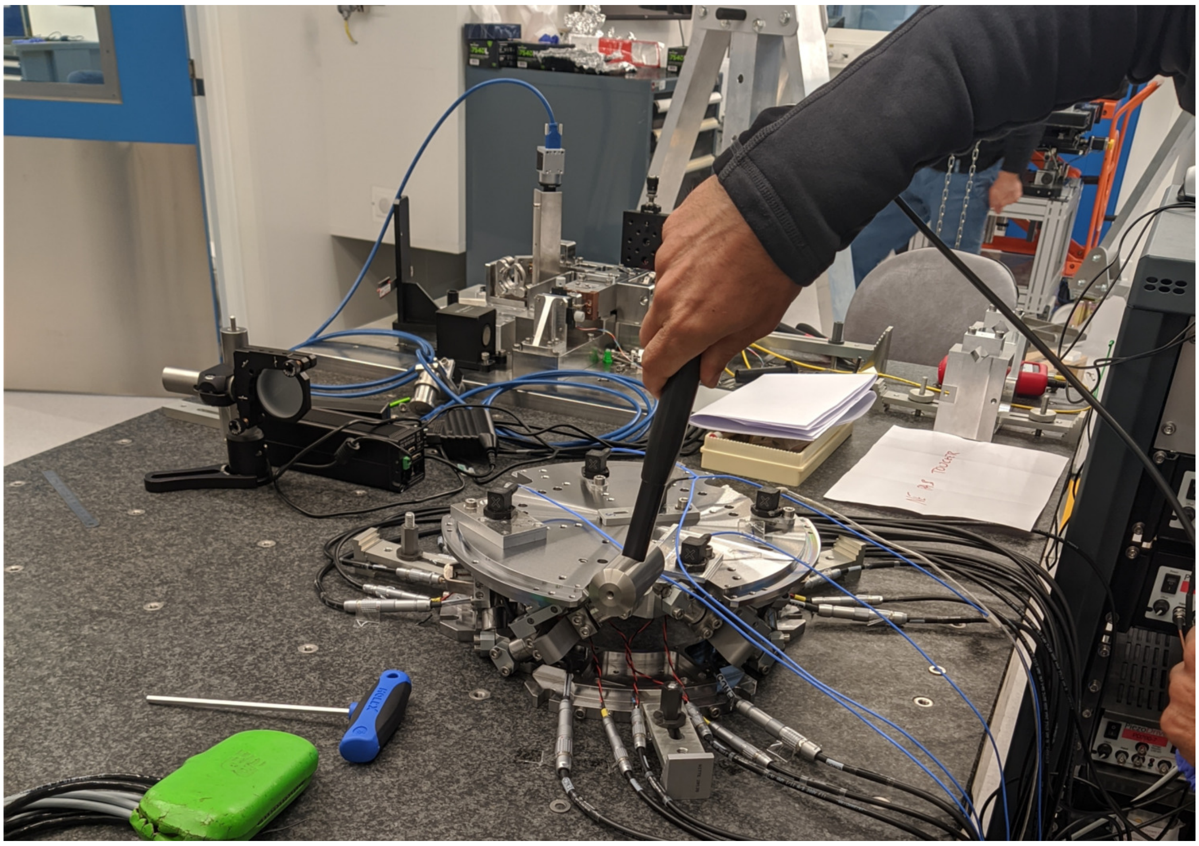


Figure 1.25: Example of an excitation using an instrumented hammer

estimated. Finally, in Section 1.4.3, the measured compliance is compare with the estimated one from the Simscape model.

1.4.1 Obtained Mode Shapes

We can observe the mode shapes of the first 6 modes that are the suspension modes (the plate is behaving as a solid body) in Figure 1.26.

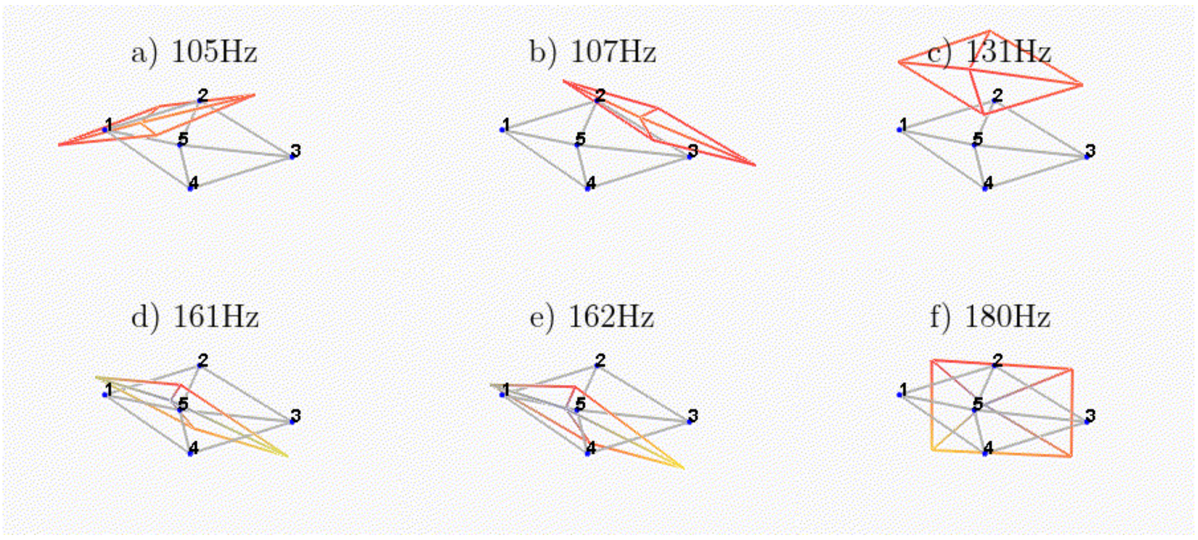


Figure 1.26: Measured mode shapes for the first six modes

Then, there is a mode at 692Hz which corresponds to a flexible mode of the top plate (Figure 1.27).

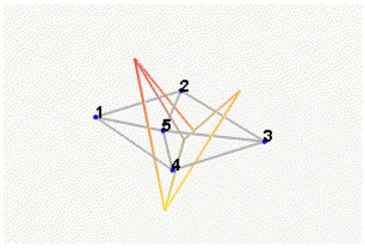


Figure 1.27: First flexible mode at 692Hz

The obtained modes are summarized in Table 1.1.

1.4.2 Nano-Hexapod Compliance - Effect of IFF

In this section, we wish to estimated the effectiveness of the IFF strategy concerning the compliance.

The top plate is excited vertically using the instrumented hammer two times:

1. no control loop is used

Table 1.1: Description of the identified modes

Mode	Freq. [Hz]	Description
1	105	Suspension Mode: Y-translation
2	107	Suspension Mode: X-translation
3	131	Suspension Mode: Z-translation
4	161	Suspension Mode: Y-tilt
5	162	Suspension Mode: X-tilt
6	180	Suspension Mode: Z-rotation
7	692	(flexible) Membrane mode of the top platform

2. decentralized IFF is used

The data is loaded.

Matlab
```
frf_ol  = load('Measurement_Z_axis.mat'); % Open-Loop
frf_iff = load('Measurement_Z_axis_damped.mat'); % IFF
```

The mean vertical motion of the top platform is computed by averaging all 5 accelerometers.

Matlab
```
%% Multiply by 10 (gain in m/s^2/V) and divide by 5 (number of accelerometers)
d_frf_ol = 10/5*(frf_ol.FFT1_H1_4_1_RMS_Y_Mod + frf_ol.FFT1_H1_7_1_RMS_Y_Mod + frf_ol.FFT1_H1_10_1_RMS_Y_Mod +
↪  frf_ol.FFT1_H1_13_1_RMS_Y_Mod + frf_ol.FFT1_H1_16_1_RMS_Y_Mod)./(2*pi*frf_ol.FFT1_H1_16_1_RMS_X_Val).^2;
d_frf_iff = 10/5*(frf_iff.FFT1_H1_4_1_RMS_Y_Mod + frf_iff.FFT1_H1_7_1_RMS_Y_Mod + frf_iff.FFT1_H1_10_1_RMS_Y_Mod +
↪  frf_iff.FFT1_H1_13_1_RMS_Y_Mod + frf_iff.FFT1_H1_16_1_RMS_Y_Mod)./(2*pi*frf_iff.FFT1_H1_16_1_RMS_X_Val).^2;
```

The vertical compliance (magnitude of the transfer function from a vertical force applied on the top plate to the vertical motion of the top plate) is shown in Figure 1.28.

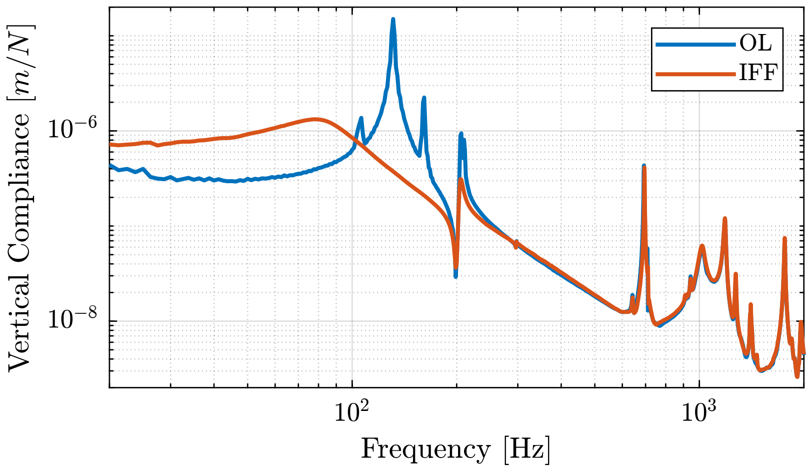


Figure 1.28: Measured vertical compliance with and without IFF

Important

From Figure 1.28, it is clear that the IFF control strategy is very effective in damping the suspensions modes of the nano-hexapod. It also has the effect of (slightly) degrading the vertical compliance at low frequency.
It also seems some damping can be added to the modes at around 205Hz which are flexible modes of the struts.

1.4.3 Comparison with the Simscape Model

Let's now compare the measured vertical compliance with the vertical compliance as estimated from the Simscape model.

The transfer function from a vertical external force to the absolute motion of the top platform is identified (with and without IFF) using the Simscape model. The comparison is done in Figure 1.29. Again, the model is quite accurate!

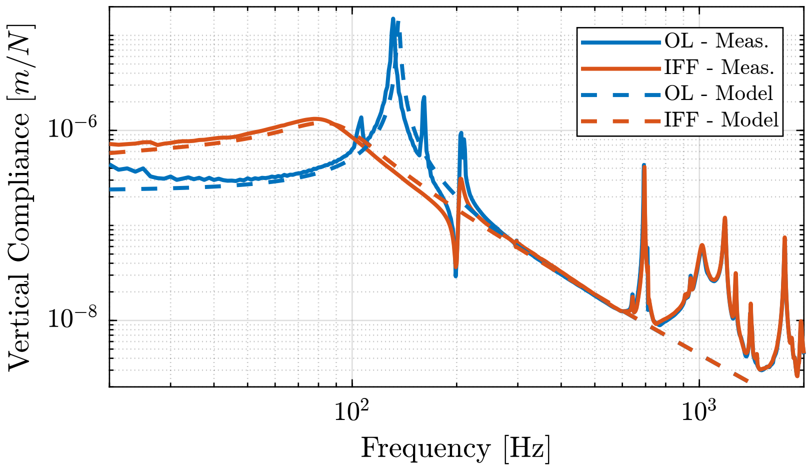


Figure 1.29: Measured vertical compliance with and without IFF

1.5 Conclusion

Important

From the previous analysis, several conclusions can be drawn:

- Decentralized IFF is very effective in damping the "suspension" modes of the nano-hexapod (Figure 1.19)
- Decentralized IFF does not damp the "spurious" modes of the struts nor the flexible modes of the top plate (Figure 1.19)
- Even though the Simscape model and the experimentally measured FRF are in good agreement (Figures 1.22 and 1.23), the obtain dynamics from the control inputs $\boldsymbol{u}$ and the

encoders $d\boldsymbol{\mathcal{L}}_m$ is very difficult to control

Therefore, in the following sections, the encoders will be fixed to the plates. The goal is to be less sensitive to the flexible modes of the struts.

2 Encoders fixed to the plates - Dynamics

In this section, the encoders are fixed to the plates rather than to the struts as shown in Figure 2.1.

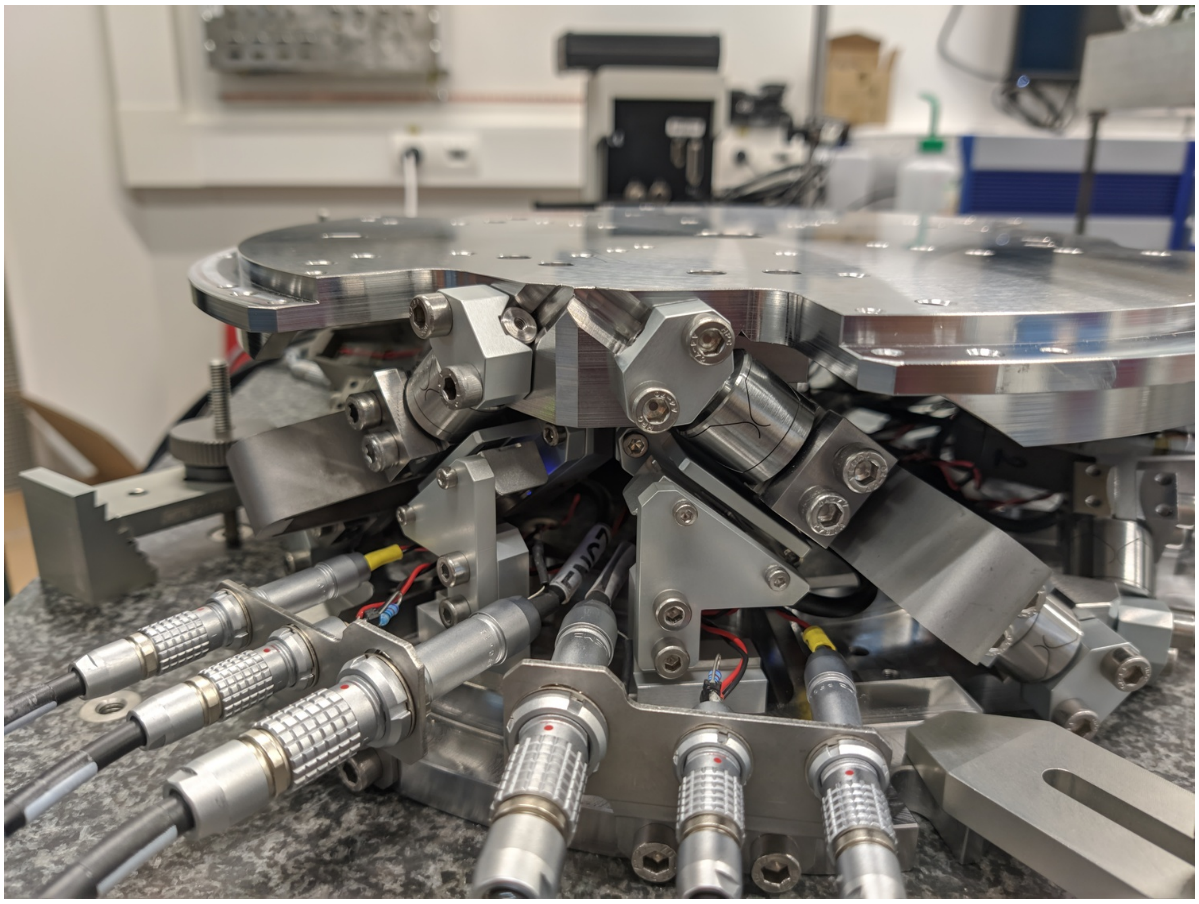


Figure 2.1: Nano-Hexapod with encoders fixed to the struts

It is structured as follow:

- Section 2.1: The dynamics of the nano-hexapod is identified.
- Section 2.2: The identified dynamics is compared with the Simscape model.
- Section 2.3: The Integral Force Feedback (IFF) control strategy is applied and the dynamics of the damped nano-hexapod is identified and compare with the Simscape model.

2.1 Identification of the dynamics

In this section, the dynamics of the nano-hexapod with the encoders fixed to the plates is identified.

First, the measurement data are loaded in Section 2.1.1, then the transfer function matrix from the actuators to the encoders are estimated in Section 2.1.2. Finally, the transfer function matrix from the actuators to the force sensors is estimated in Section 2.1.3.

2.1.1 Data Loading and Spectral Analysis Setup

The actuators are excited one by one using a low pass filtered white noise. For each excitation, the 6 force sensors and 6 encoders are measured and saved.

Matlab
```
%% Load Identification Data
meas_data_lf = {};

for i = 1:6
    meas_data_lf(i) = {load(sprintf('mat/frf_exc_strut_%i_enc_plates_noise.mat', i), 't', 'Va', 'Vs', 'de')};
end
```

2.1.2 Transfer function from Actuator to Encoder

Let's compute the coherence from the excitation voltage $\boldsymbol{u}$ and the displacement $d\boldsymbol{\mathcal{L}}_m$ as measured by the encoders.

Matlab
```
%% Coherence
coh_dvf = zeros(length(f), 6, 6);

for i = 1:6
    coh_dvf(:, :, i) = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
end
```

The obtained coherence shown in Figure 2.2 is quite good up to 400Hz.

Then the 6x6 transfer function matrix is estimated.

Matlab
```
%% DVF Plant (transfer function from u to dLm)
G_dvf = zeros(length(f), 6, 6);

for i = 1:6
    G_dvf(:,:,i) = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
end
```

The diagonal and off-diagonal terms of this transfer function matrix are shown in Figure 2.3.

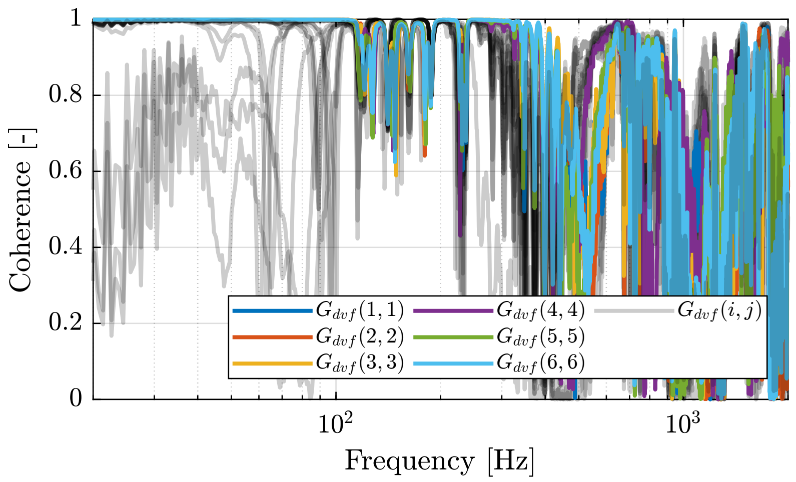


Figure 2.2: Obtained coherence for the DVF plant

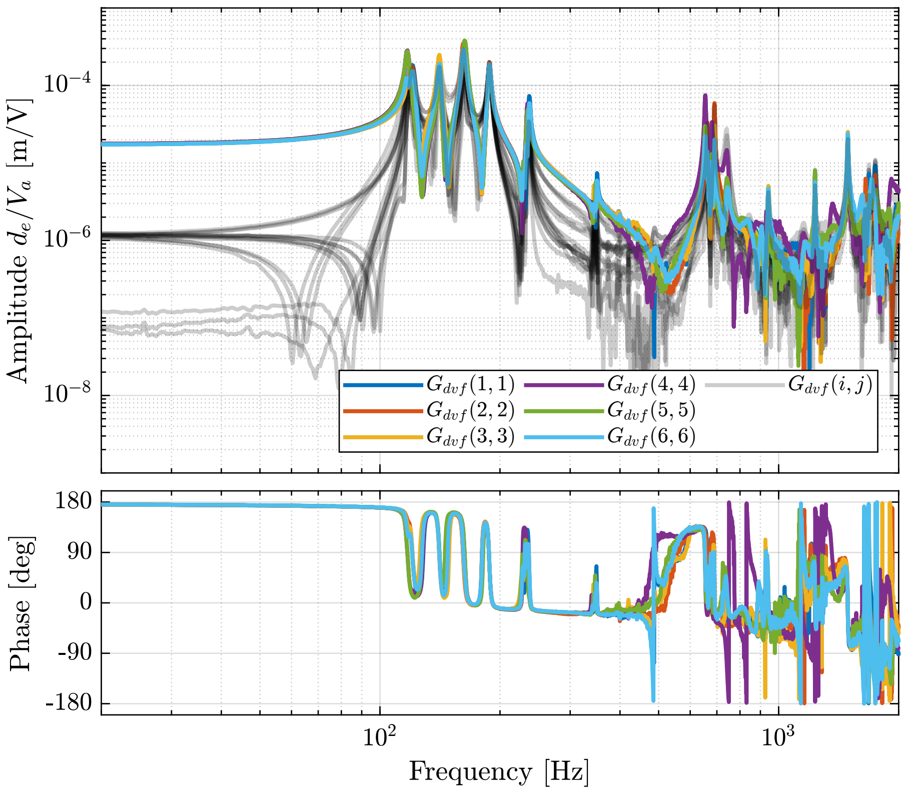


Figure 2.3: Measured FRF for the DVF plant

Important

From Figure 2.3, we can draw few conclusions on the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ when the encoders are fixed to the plates:

- the decoupling is rather good at low frequency (below the first suspension mode). The low frequency gain is constant for the off diagonal terms, whereas when the encoders where fixed to the struts, the low frequency gain of the off-diagonal terms were going to zero (Figure 1.2).
- the flexible modes of the struts at 226Hz and 337Hz are indeed shown in the transfer functions, but their amplitudes are rather low.
- the diagonal terms have alternating poles and zeros up to at least 600Hz: the flexible modes of the struts are not affecting the alternating pole/zero pattern. This what not the case when the encoders were fixed to the struts (Figure 1.2).

2.1.3 Transfer function from Actuator to Force Sensor

Let's now compute the coherence from the excitation voltage $\boldsymbol{u}$ and the voltage $\boldsymbol{\tau}_m$ generated by the Force senors.

```matlab
%% Coherence for the IFF plant
coh_iff = zeros(length(f), 6, 6);

for i = 1:6
    coh_iff(:,:,i) = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
end
```

The coherence is shown in Figure 2.4, and is very good for from 10Hz up to 2kHz.

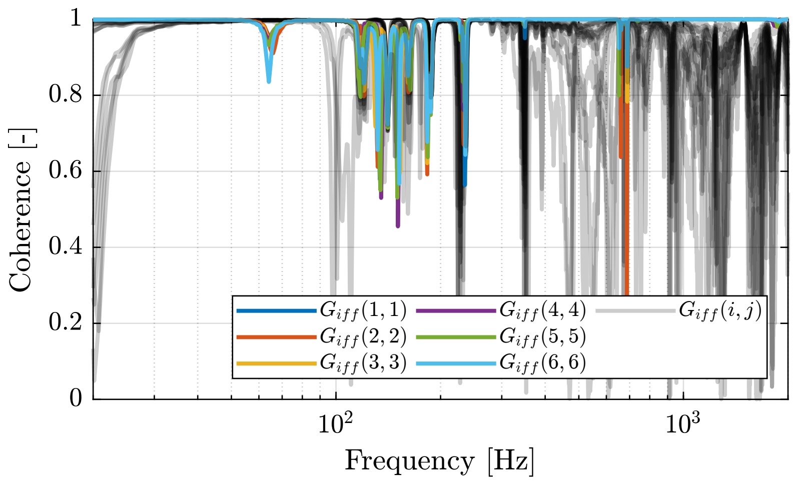


Figure 2.4: Obtained coherence for the IFF plant

Then the 6x6 transfer function matrix is estimated.

Matlab
```
%% IFF Plant
G_iff = zeros(length(f), 6, 6);

for i = 1:6
    G_iff(:,:,i) = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
end
```

The bode plot of the diagonal and off-diagonal terms are shown in Figure 2.5.

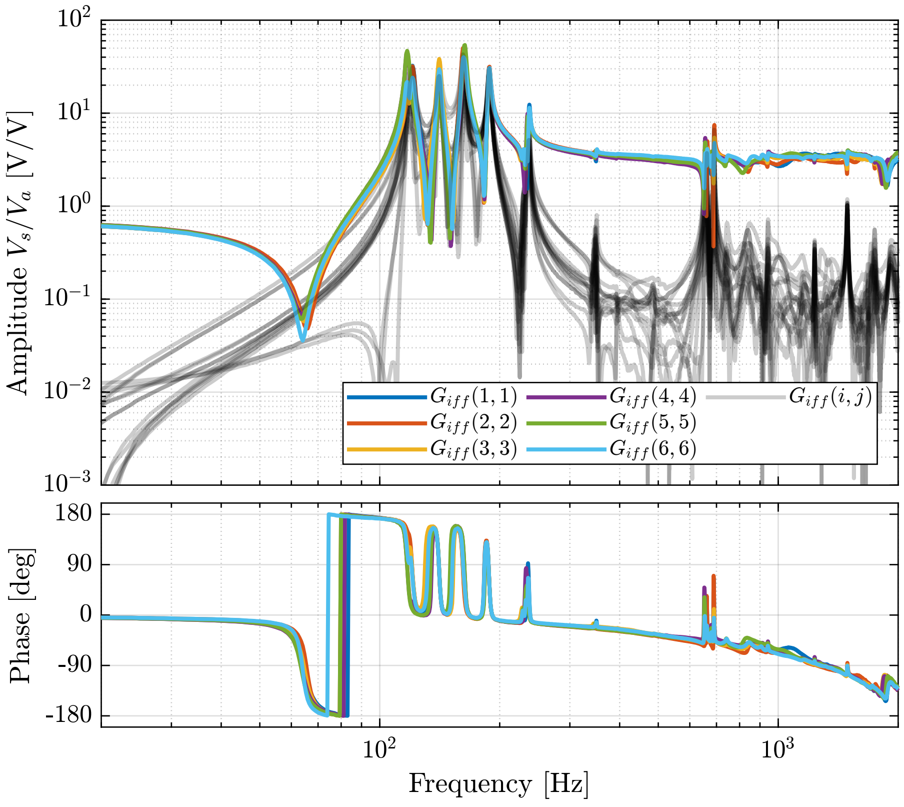


Figure 2.5: Measured FRF for the IFF plant

Important

It is shown in Figure 2.6 that:

- The IFF plant has alternating poles and zeros
- The first flexible mode of the struts as 235Hz is appearing, and therefore is should be possible to add some damping to this mode using IFF
- The decoupling is quite good at low frequency (below the first model) as well as high frequency (above the last suspension mode, except near the flexible modes of the top plate)

2.1.4 Save Identified Plants

The identified dynamics is saved for further use.

```matlab
save('mat/identified_plants_enc_plates.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

2.2 Comparison with the Simscape Model

In this section, the measured dynamics done in Section 2.1 is compared with the dynamics estimated from the Simscape model.

2.2.1 Identification Setup

The nano-hexapod is initialized with the APA taken as flexible models.

```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible');
```

2.2.2 Dynamics from Actuator to Force Sensors

Then the transfer function from $\boldsymbol{u}$ to $\boldsymbol{\tau}_m$ is identified using the Simscape model.

```matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

The identified dynamics is compared with the measured FRF:

- Figure 2.6: the individual transfer function from u_1 (the DAC voltage for the first actuator) to the force sensors of all 6 struts are compared
- Figure 2.7: all the diagonal elements are compared
- Figure 2.8: all the off-diagonal elements are compared

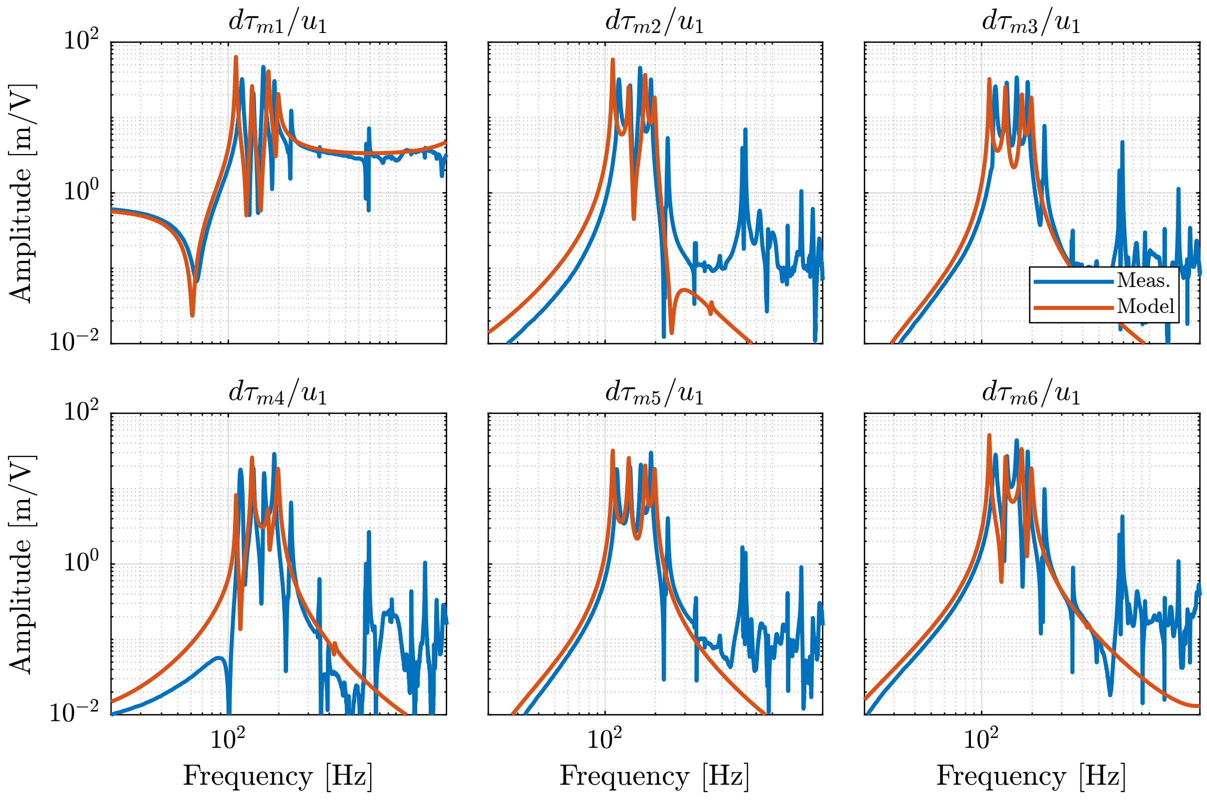


Figure 2.6: IFF Plant for the first actuator input and all the force senosrs

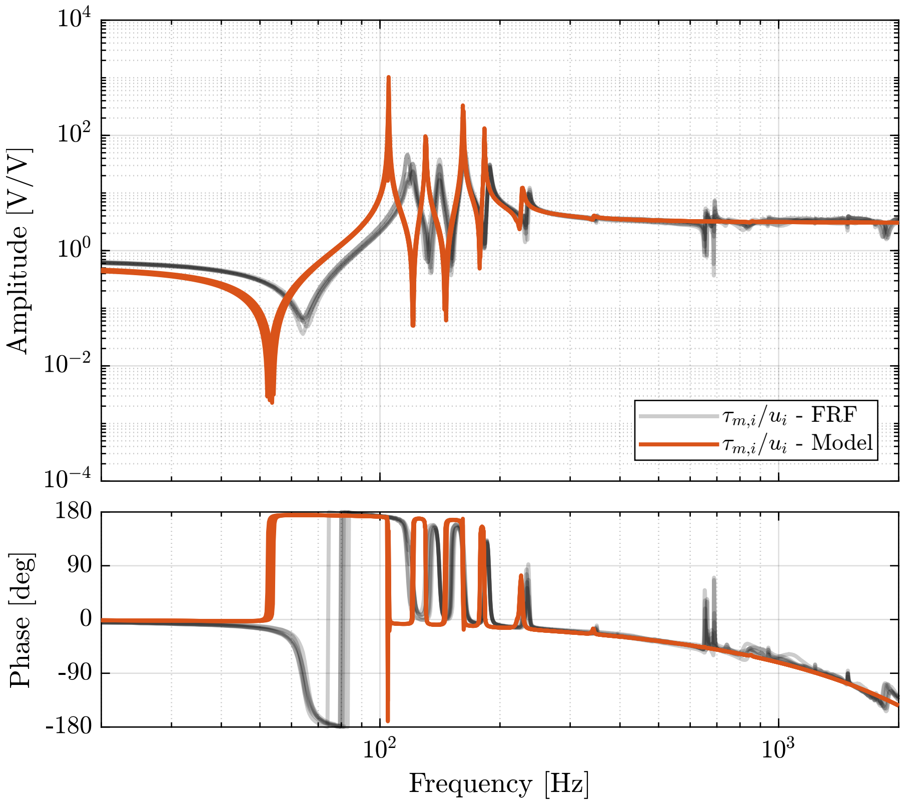


Figure 2.7: Diagonal elements of the IFF Plant

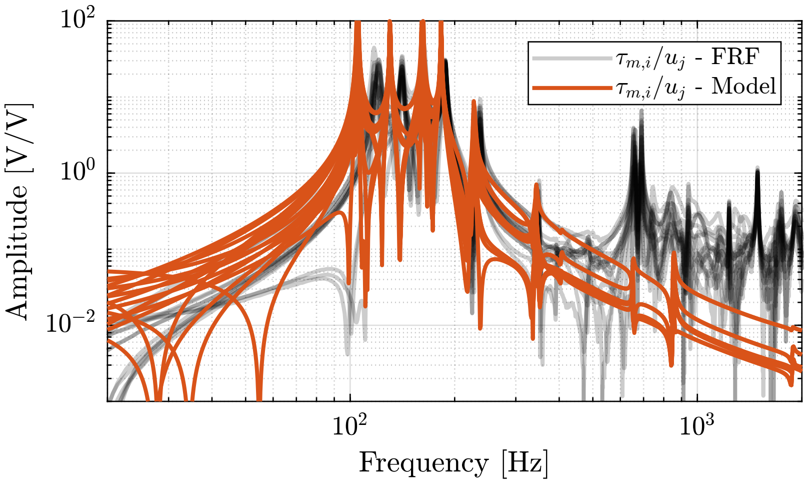


Figure 2.8: Off diagonal elements of the IFF Plant

2.2.3 Dynamics from Actuator to Encoder

Now, the dynamics from the DAC voltage $\bm{u}$ to the encoders $d\bm{\mathcal{L}}_m$ is estimated using the Simscape model.

```
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Encoders

Gdvf = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

The identified dynamics is compared with the measured FRF:

- Figure 2.9: the individual transfer function from u_3 (the DAC voltage for the actuator number 3) to the six encoders
- Figure 2.10: all the diagonal elements are compared
- Figure 2.11: all the off-diagonal elements are compared

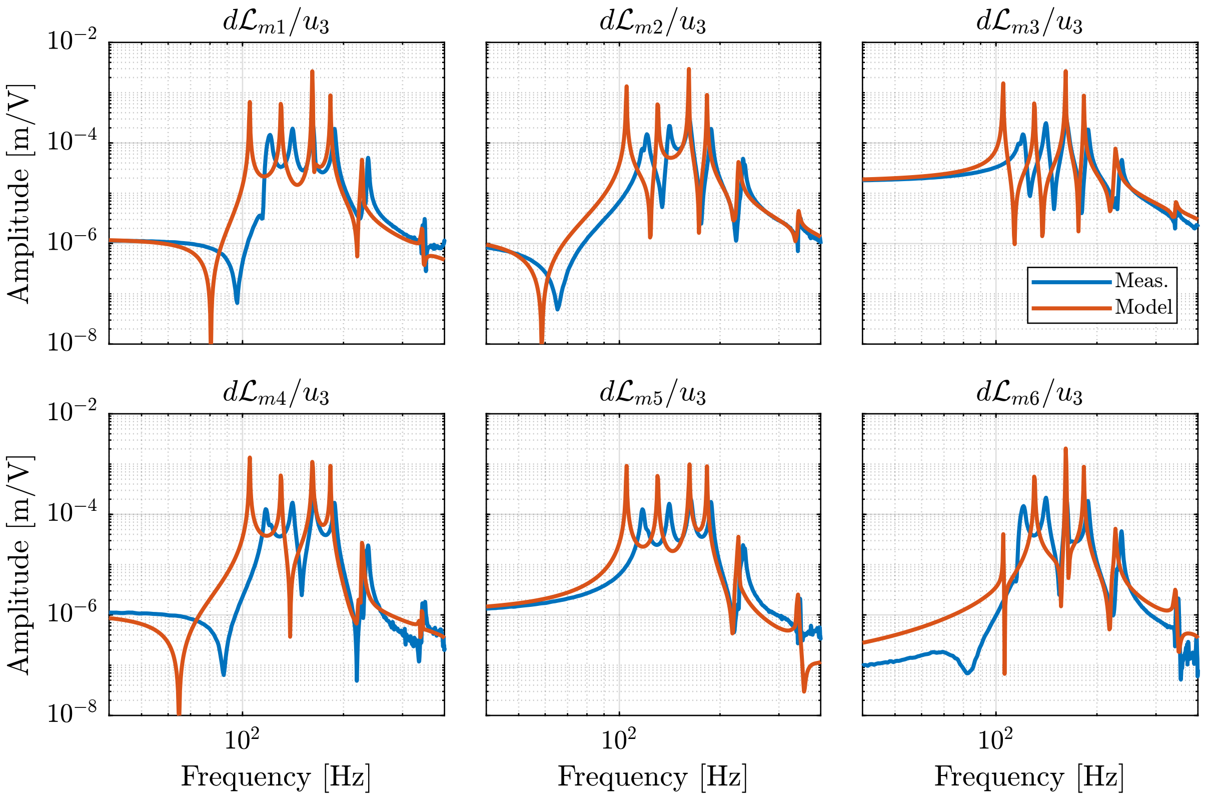


Figure 2.9: DVF Plant for the first actuator input and all the encoders

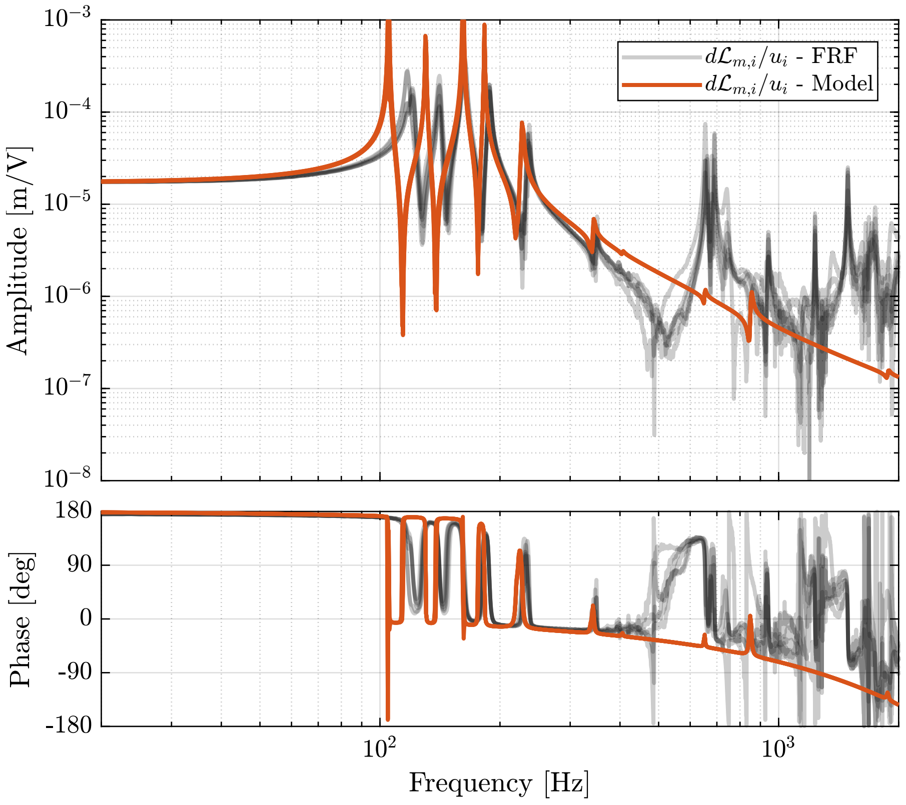


Figure 2.10: Diagonal elements of the DVF Plant

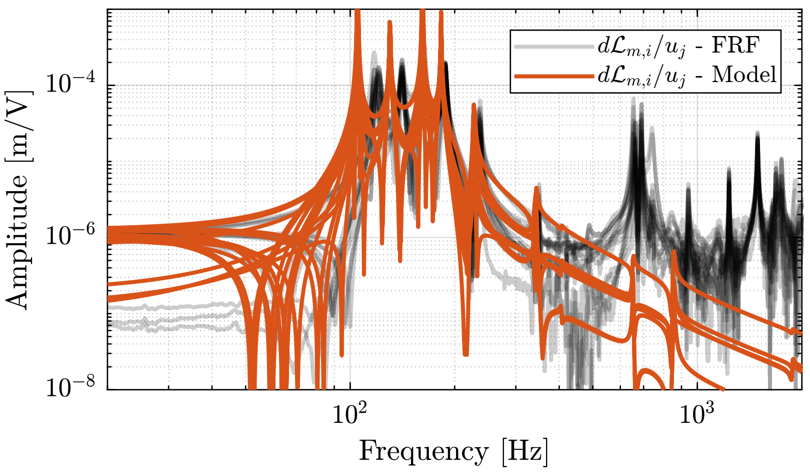


Figure 2.11: Off diagonal elements of the DVF Plant

2.2.4 Conclusion

Important

The Simscape model is quite accurate for the transfer function matrices from $\boldsymbol{u}$ to $\boldsymbol{\tau}_m$ and from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ except at frequencies of the flexible modes of the top-plate. The Simscape model can therefore be used to develop the control strategies.

2.3 Integral Force Feedback

In this section, the Integral Force Feedback (IFF) control strategy is applied to the nano-hexapod in order to add damping to the suspension modes.

The control architecture is shown in Figure 2.12:

- $\boldsymbol{\tau}_m$ is the measured voltage of the 6 force sensors
- $\boldsymbol{K}_{\text{IFF}}$ is the 6×6 diagonal controller
- $\boldsymbol{u}$ is the plant input (voltage generated by the 6 DACs)
- $\boldsymbol{u}'$ is the new plant inputs with added damping

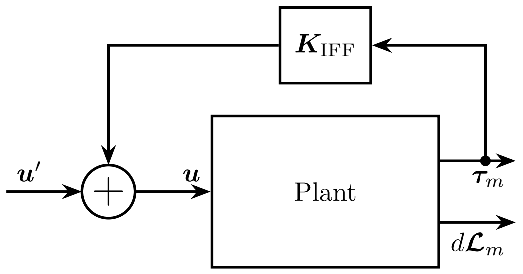


Figure 2.12: Integral Force Feedback Strategy

- Section 2.3.1

2.3.1 Effect of IFF on the plant - Simscape Model

The nano-hexapod is initialized with flexible APA and the encoders fixed to the struts.

```
Matlab
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible');
```

The same controller as the one developed when the encoder were fixed to the struts is used.

```
Matlab
%% Optimal IFF controller
load('Kiff.mat', 'Kiff')
```

The transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ is identified.

```
Matlab
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Plate Displacement (encoder)
```

First in Open-Loop:

```
Matlab
%% Transfer function from u to dL (open-loop)
Gd_ol = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

And then with the IFF controller:

```
Matlab
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'controller_type', 'iff');

%% Transfer function from u to dL (IFF)
Gd_iff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

It is first verified that the system is stable:

```
Matlab
isstable(Gd_iff)
```

```
Results
1
```

The diagonal and off-diagonal terms of the 6×6 transfer function matrices identified are compared in Figure 2.13. It is shown, as was the case when the encoders were fixed to the struts, that the IFF control strategy is very effective in damping the suspension modes of the nano-hexapod.

2.3.2 Effect of IFF on the plant - FRF

The IFF control strategy is experimentally implemented. The (damped) transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ is experimentally identified.

The identification data are loaded:

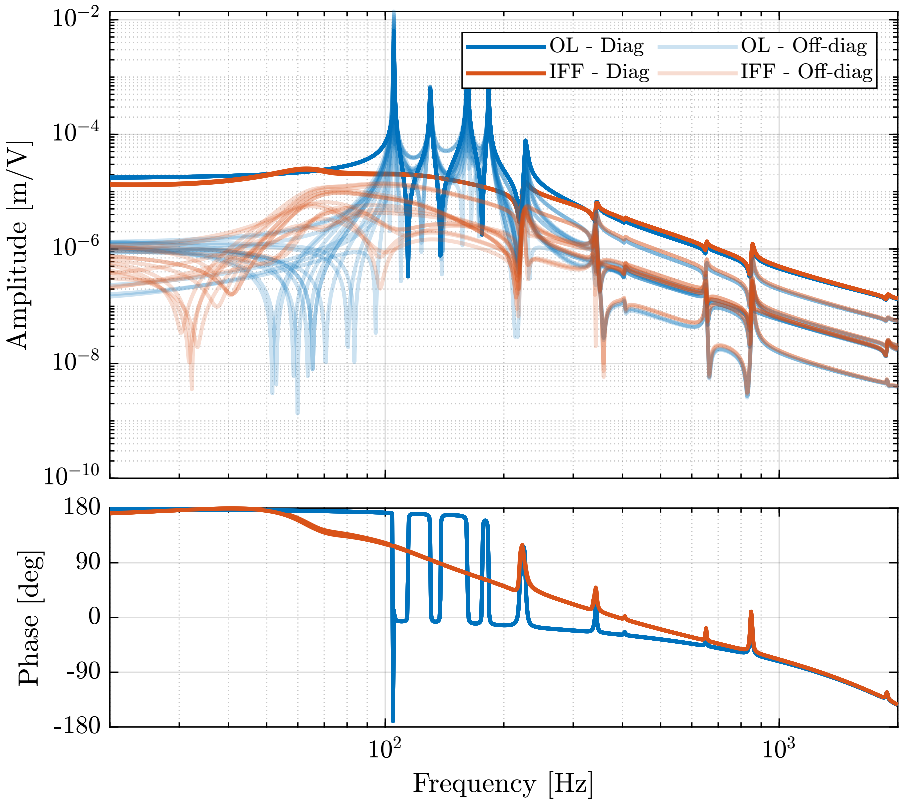


Figure 2.13: Effect of the IFF control strategy on the transfer function from $\boldsymbol{\tau}$ to $d\boldsymbol{\mathcal{L}}_m$

Matlab

```matlab
%% Load Identification Data
meas_iff_plates = {};

for i = 1:6
    meas_iff_plates(i) = {load(sprintf('mat/frf_exc_iff_strut_%i_enc_plates_noise.mat', i), 't', 'Va', 'Vs', 'de', 'u')};
end
```

And the parameters used for the transfer function estimation are defined below.

Matlab

```matlab
% Sampling Time [s]
Ts = (meas_iff_plates{1}.t(end) - (meas_iff_plates{1}.t(1)))/(length(meas_iff_plates{1}.t)-1);

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_iff_plates{1}.Va, meas_iff_plates{1}.de, win, [], [], 1/Ts);
```

The estimation is performed using the `tfestimate` command.

Matlab

```matlab
%% Estimation of the transfer function matrix from u to dL when IFF is applied
G_enc_iff_opt = zeros(length(f), 6, 6);

for i = 1:6
    G_enc_iff_opt(:,:,i) = tfestimate(meas_iff_plates{i}.Va, meas_iff_plates{i}.de, win, [], [], 1/Ts);
end
```

The obtained diagonal and off-diagonal elements of the transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ are shown in Figure 2.14 both without and with IFF.

Important

As was predicted with the Simscape model, the IFF control strategy is very effective in damping the suspension modes of the nano-hexapod. Little damping is also applied on the first flexible mode of the strut at 235Hz. However, no damping is applied on other modes, such as the flexible modes of the top plate.

2.3.3 Comparison of the measured FRF and the Simscape model

Let's now compare the obtained damped plants obtained experimentally with the one extracted from Simscape:

- Figure 2.15: the individual transfer function from u_1' to the six encoders are comapred
- Figure 2.16: all the diagonal elements are compared
- Figure 2.17: all the off-diagonal elements are compared

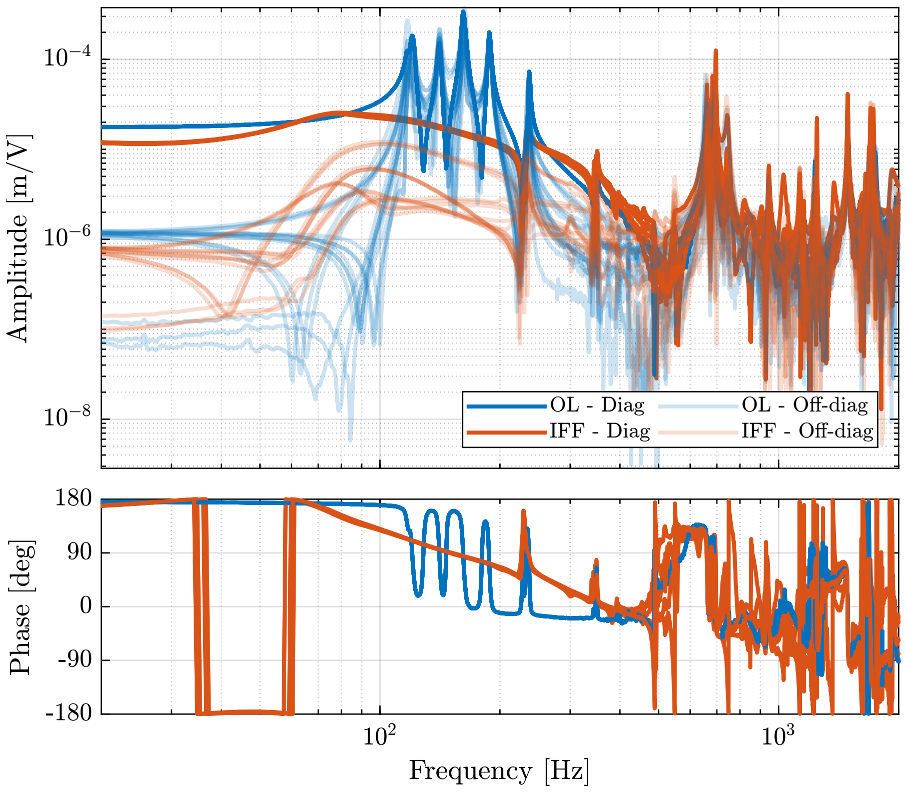


Figure 2.14: Effect of the IFF control strategy on the transfer function from $\boldsymbol{\tau}$ to $d\boldsymbol{\mathcal{L}}_m$

Important

From Figures 2.16 and 2.17, it is clear that the Simscape model very well represents the dynamics of the nano-hexapod. This is true to around 400Hz, then the dynamics depends on the flexible modes of the top plate which are not modelled.

2.3.4 Save Damped Plant

The experimentally identified plant is saved for further use.

```matlab
save('matlab/mat/damped_plant_enc_plates.mat', 'f', 'Ts', 'G_enc_iff_opt')
```

```matlab
save('mat/damped_plant_enc_plates.mat', 'f', 'Ts', 'G_enc_iff_opt')
```

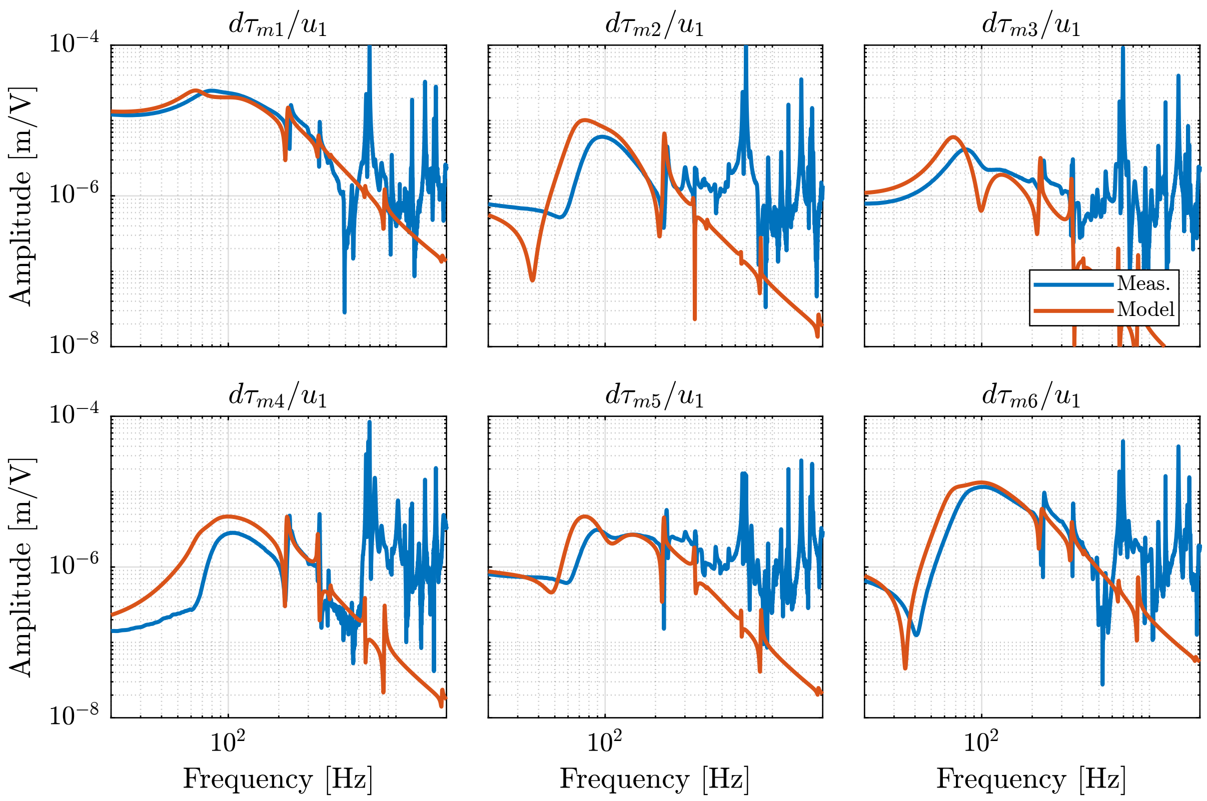


Figure 2.15: FRF from one actuator to all the encoders when the plant is damped using IFF

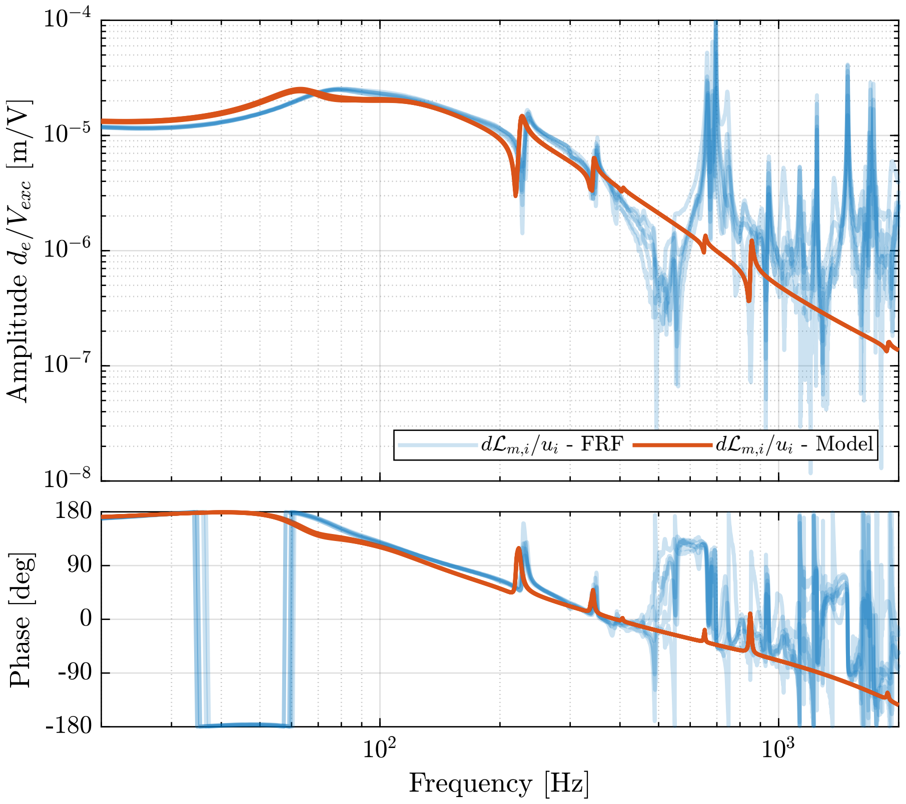


Figure 2.16: Comparison of the diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

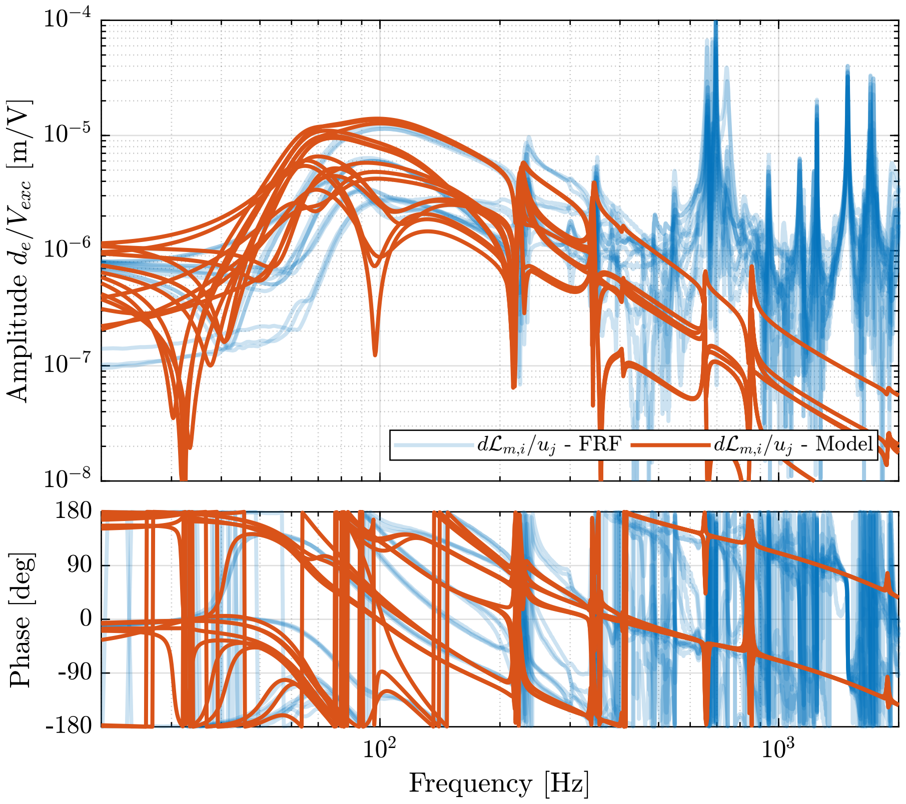


Figure 2.17: Comparison of the off-diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

2.4 Conclusion

Important

In this section, the dynamics of the nano-hexapod with the encoders fixed to the plates is studied. It has been found that:

- The measured dynamics is in agreement with the dynamics of the simscape model, up to the flexible modes of the top plate. See figures 2.7 and 2.8 for the transfer function to the force sensors and Figures 2.10 and 2.11for the transfer functions to the encoders
- The Integral Force Feedback strategy is very effective in damping the suspension modes of the nano-hexapod (Figure 2.14).
- The transfer function from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ show nice dynamical properties and is a much better candidate for the high-authority-control than when the encoders were fixed to the struts. At least up to the flexible modes of the top plate, the diagonal elements of the transfer function matrix have alternating poles and zeros, and the phase is moving smoothly. Only the flexible modes of the top plates seems to be problematic for control.

3 Decentralized High Authority Control with Integral Force Feedback

In this section is studied the HAC-LAC architecture for the Nano-Hexapod. More precisely:

- The LAC control is a decentralized force feedback as studied in Section 2.3
- The HAC control is a decentralized controller working in the frame of the struts

The corresponding control architecture is shown in Figure 3.1 with:

- $\boldsymbol{r}_{\mathcal{X}_n}$: the 6×1 reference signal in the cartesian frame
- $\boldsymbol{r}_{d\mathcal{L}}$: the 6×1 reference signal transformed in the frame of the struts thanks to the inverse kinematic
- $\boldsymbol{\epsilon}_{d\mathcal{L}}$: the 6×1 length error of the 6 struts
- $\boldsymbol{u}'$: input of the damped plant
- $\boldsymbol{u}$: generated DAC voltages
- $\boldsymbol{\tau}_m$: measured force sensors
- $d\boldsymbol{\mathcal{L}}_m$: measured displacement of the struts by the encoders

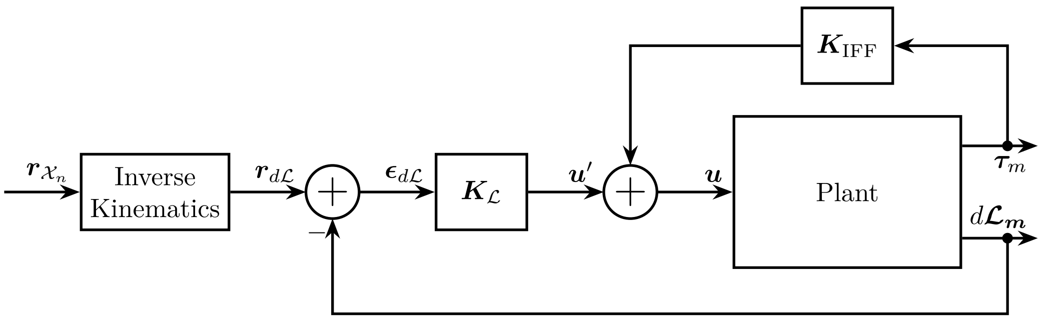

$\boldsymbol{r}_{\mathcal{X}_n}$
Inverse Kinematics
$\boldsymbol{r}_{d\mathcal{L}}$
$-$
$\boldsymbol{\epsilon}_{d\mathcal{L}}$
$\boldsymbol{K}_{\mathcal{L}}$
$\boldsymbol{u}'$
$\boldsymbol{K}_{\text{IFF}}$
$\boldsymbol{u}$
Plant
$\boldsymbol{\tau}_m$
$d\boldsymbol{\mathcal{L}}_m$

Figure 3.1: HAC-LAC: IFF + Control in the frame of the legs

- Section 3.1: the decentralized high authority controller is tuned using the Simscape model
- Section 3.3: the controller is implemented and tested experimentally
- Section 3.2: some reference tracking tests are performed

3.1 High Authority Controller

In this section, the decentralized high authority controller $\boldsymbol{K}_\mathcal{L}$ is first tuned using the Simscape model.

3.1.1 Simscape Model

First initialized the nano-hexapod with a flexible APA model and with the IFF control strategy.

Matlab
```matlab
%% Initialize the Simscape model
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'controller_type', 'iff');
```

Then the controller is loaded

Matlab
```matlab
%% Load the decentralized IFF controller
load('Kiff.mat', 'Kiff')
```

The inputs and outputs for the transfer function estimation are defined.

Matlab
```matlab
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Plate Displacement (encoder)
```

And the plant from $\boldsymbol{u}'$ to $d\boldsymbol{\mathcal{L}}_m$ is identified and the bode plot of its diagonal terms are shown in Figure 3.2.

Matlab
```matlab
%% Identified of the damped TF from u' to dL
Gd_iff_opt = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

3.1.2 HAC Controller

Let's first try to design a first decentralized controller with:

- a bandwidth of 100Hz
- sufficient phase margin
- simple and understandable components

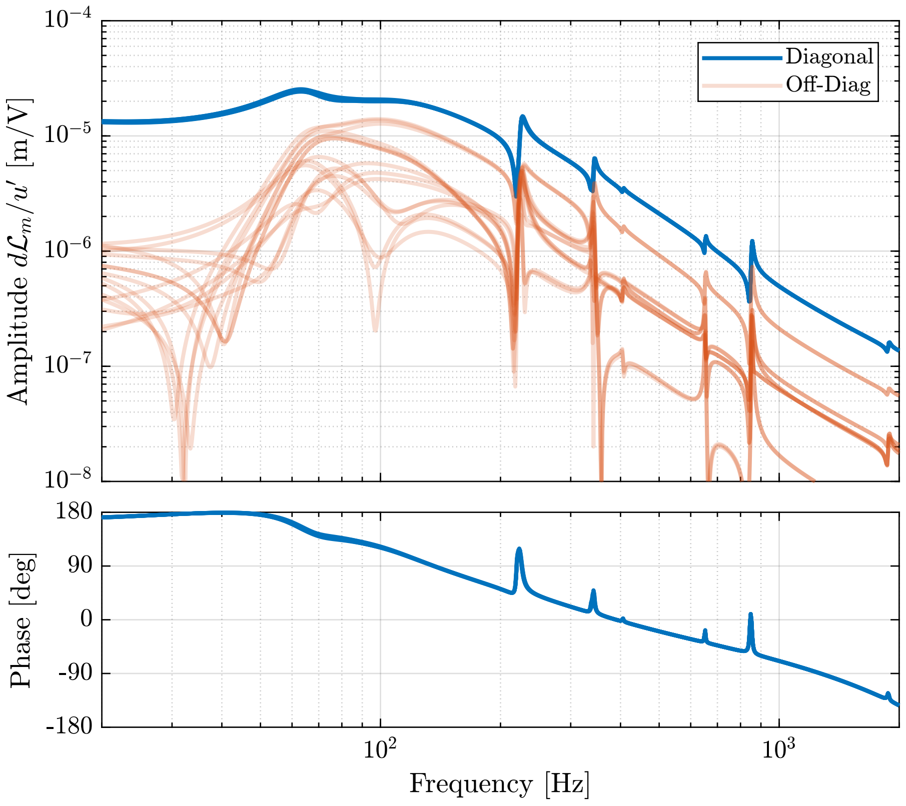


Figure 3.2: Transfer functions from u to $d\mathcal{L}_m$ with IFF (diagonal and off-diagonal elements)

After some very basic and manual loop shaping, the following controller is developed:

```matlab
%% Lead to increase phase margin
a  = 2;  % Amount of phase lead / width of the phase lead / high frequency gain
wc = 2*pi*100; % Frequency with the maximum phase lead [rad/s]

H_lead = (1 + s/(wc/sqrt(a)))/(1 + s/(wc*sqrt(a)));

%% Low Pass filter to increase robustness
H_lpf = 1/(1 + s/2/pi/200);

%% Notch at the top-plate resonance
gm = 0.02;
xi = 0.3;
wn = 2*pi*700;

H_notch = (s^2 + 2*gm*xi*wn*s + wn^2)/(s^2 + 2*xi*wn*s + wn^2);

%% Decentralized HAC
Khac_iff_struts = -(1/(2.87e-5)) * ... % Gain
                  H_lead * ...       % Lead
                  H_notch * ...      % Notch
                  (2*pi*100/s) * ... % Integrator
                  eye(6);            % 6x6 Diagonal
```

This controller is saved for further use.

```matlab
save('mat/Khac_iff_struts.mat', 'Khac_iff_struts')
```

The Loop Gain is computed and shown in Figure 3.3.

```matlab
Lhac_iff_struts = Khac_iff_struts*Gd_iff_opt;
```

3.1.3 Verification of the Stability using the Simscape model

The HAC-IFF control strategy is implemented using Simscape.

```matlab
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'controller_type', 'hac-iff-struts');
```

We identify the closed-loop system.

```matlab
%% Identification
Gd_iff_hac_opt = linearize(mdl, io, 0.0, options);
```

And verify that it is indeed stable.

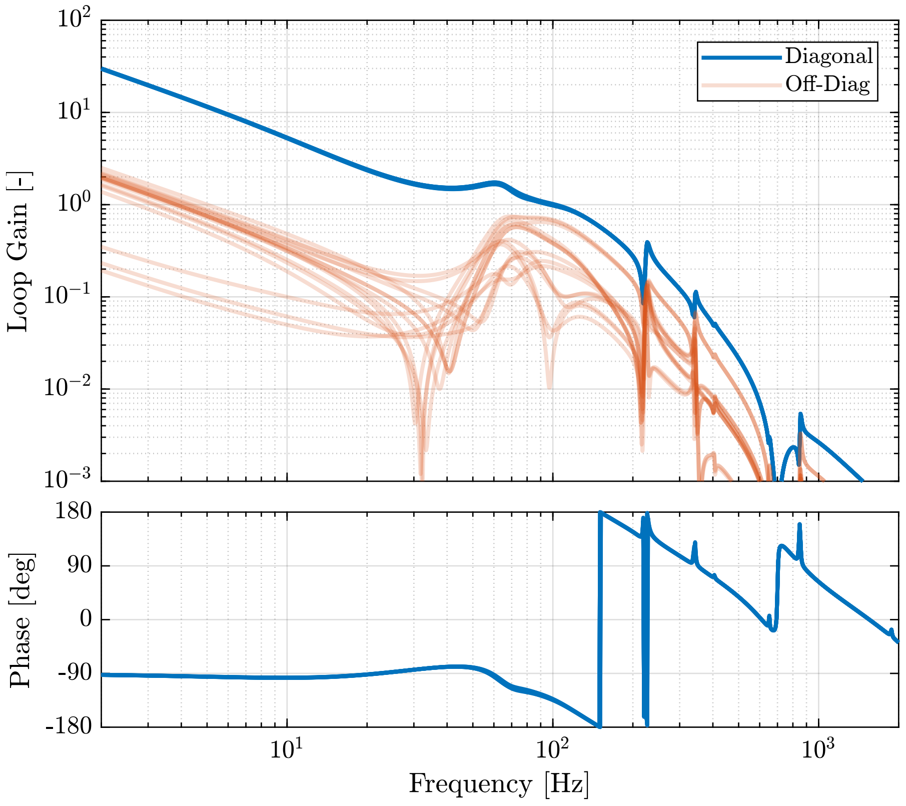


Figure 3.3: Diagonal and off-diagonal elements of the Loop gain for “HAC-IFF-Struts”

Matlab
```
%% Verify the stability
isstable(Gd_iff_hac_opt)
```

Results
```
1
```

3.1.4 Experimental Loop Gain

Now, the loop gain is estimated from the measured FRF.

Matlab
```
L_frf = zeros(size(G_enc_iff_opt));

for i = 1:size(G_enc_iff_opt, 1)
    L_frf(i, :, :) = squeeze(G_enc_iff_opt(i,:,:))*freqresp(Khac_iff_struts, f(i), 'Hz');
end
```

The bode plot of the loop gain is shown in Figure 3.4.

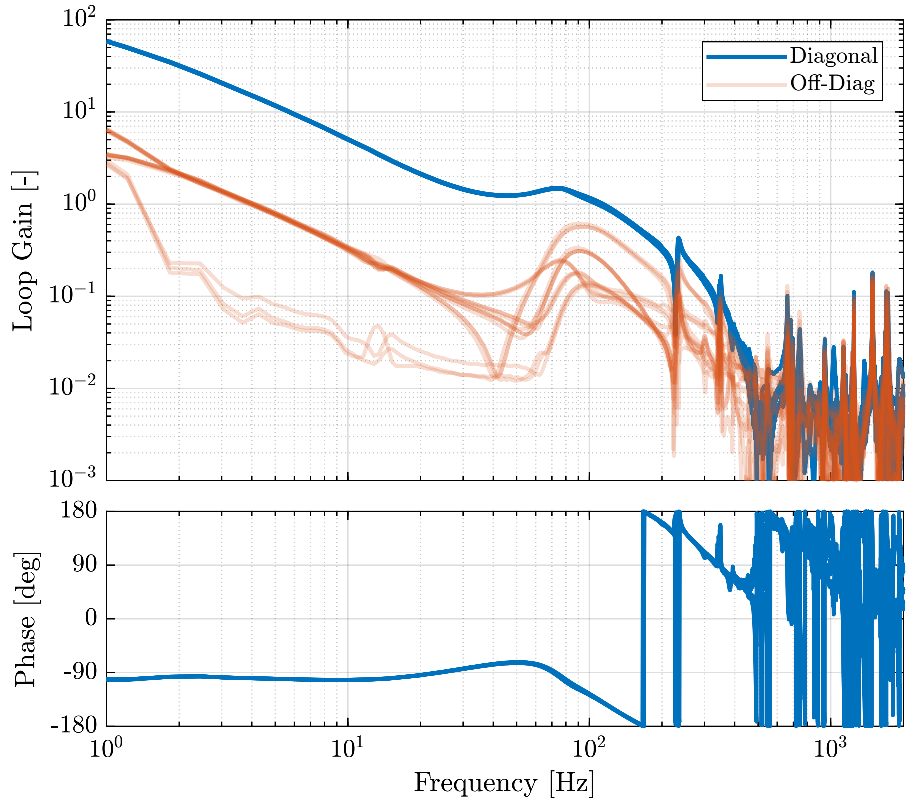


Figure 3.4: Diagonal and Off-diagonal elements of the Loop gain (experimental data)

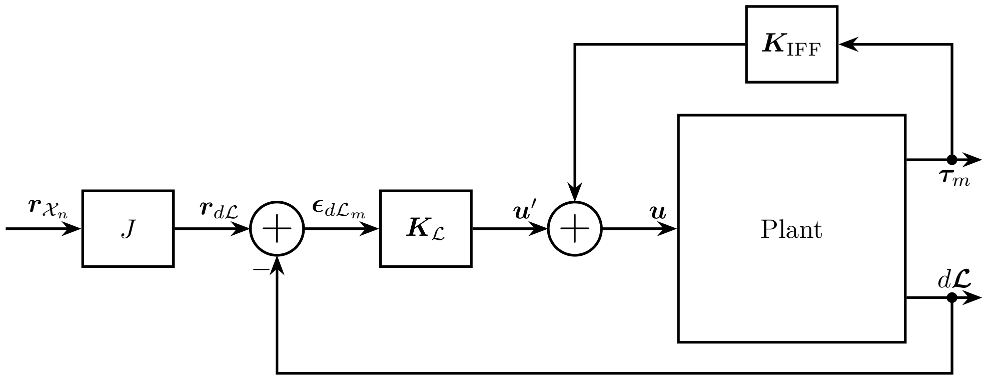
3.2 Reference Tracking - Trajectories

In this section, several trajectories representing the wanted pose (position and orientation) of the top platform with respect to the bottom platform are defined.

These trajectories will be used to test the HAC-LAC architecture.

In order to transform the wanted pose to the wanted displacement of the 6 struts, the inverse kinematic is required. As a first approximation, the Jacobian matrix can be used instead of using the full inverse kinematic equations.

Therefore, the control architecture with the input trajectory $\boldsymbol{r}_{\mathcal{X}_n}$ is shown in Figure 3.5.



$\boldsymbol{K}_{\text{IFF}}$
$\boldsymbol{r}_{\mathcal{X}_n}$
J
$\boldsymbol{r}_{d\mathcal{L}}$
$\boldsymbol{\epsilon}_{d\mathcal{L}_m}$
$\boldsymbol{K}_{\mathcal{L}}$
$\boldsymbol{u}'$
$\boldsymbol{u}$
Plant
$\boldsymbol{\tau}_m$
$d\boldsymbol{\mathcal{L}}$
$-$

Figure 3.5: HAC-LAC: IFF + Control in the frame of the legs

In the following sections, several reference trajectories are defined:

- Section 3.2.1: simple scans in the Y-Z plane
- Section 3.2.2: scans in tilt are performed
- Section 3.2.3: scans with X-Y-Z translations in order to draw the word “NASS”

3.2.1 Y-Z Scans

Generate the Scan

A function `generateYZScanTrajectory` has been developed (accessible here) in order to easily generate scans in the Y-Z plane.

For instance, the following generated trajectory is represented in Figure 3.6.

```matlab
%% Generate the Y-Z trajectory scan
Rx_yz = generateYZScanTrajectory(...
    'y_tot', 4e-6, ... % Length of Y scans [m]
    'z_tot', 8e-6, ... % Total Z distance [m]
    'n', 5, ...      % Number of Y scans
    'Ts', 1e-3, ... % Sampling Time [s]
    'ti', 2, ...     % Time to go to initial position [s]
    'tw', 0.5, ...  % Waiting time between each points [s]
    'ty', 2, ...    % Time for a scan in Y [s]
    'tz', 1);       % Time for a scan in Z [s]
```

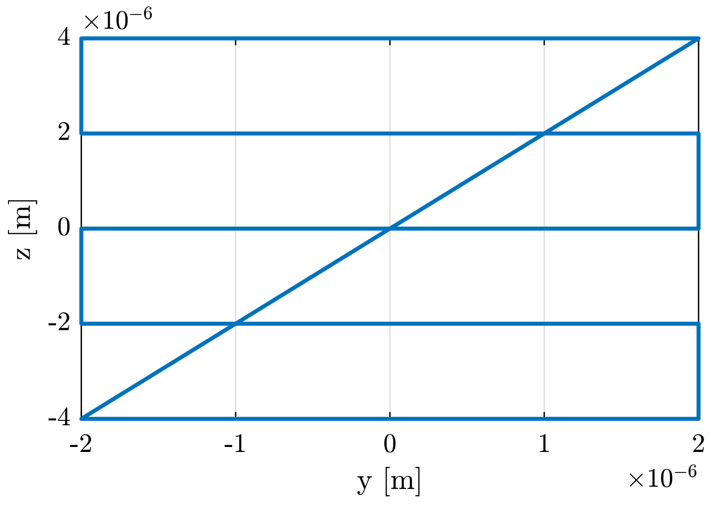


Figure 3.6: Generated scan in the Y-Z plane

The Y and Z positions as a function of time are shown in Figure 3.7.

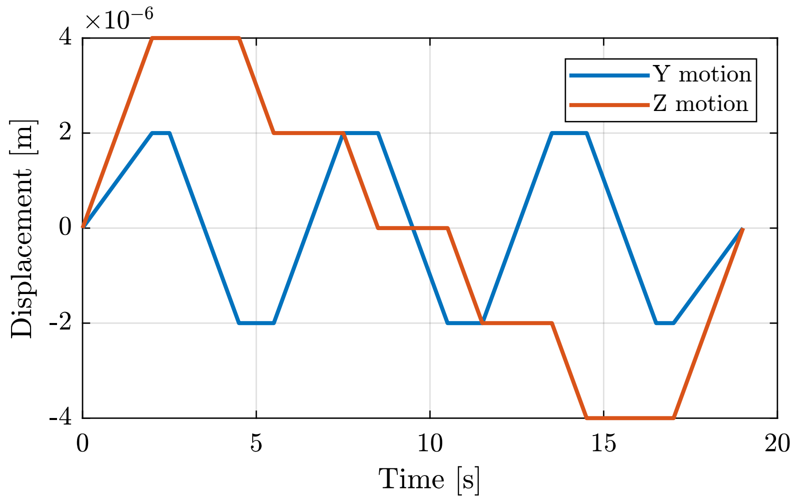


Figure 3.7: Y and Z trajectories as a function of time

Reference Signal for the Strut lengths

Using the Jacobian matrix, it is possible to compute the wanted struts lengths as a function of time:

$$\boldsymbol{r}_{d\mathcal{L}} = \boldsymbol{J}\boldsymbol{r}_{\mathcal{X}_n} \tag{3.1}$$

Matlab
```
dL_ref = [n_hexapod.geometry.J*Rx_yz(:, 2:7)']';
```

The reference signal for the strut length is shown in Figure 3.8.

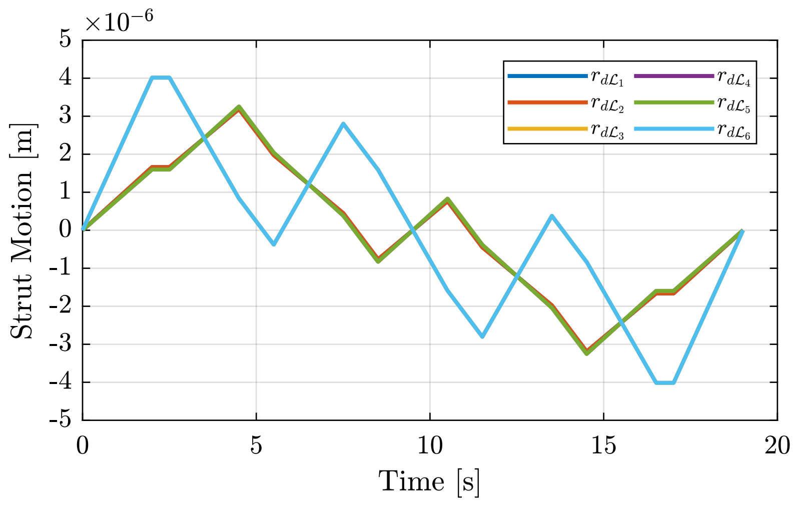


Figure 3.8: Trajectories for the 6 individual struts

Time domain simulation with 2DoF model

Before trying to follow this reference with the nano-hexapod, let's try to do it using the Simscape model.

The nano-hexapod is initialized with the APA modelled as 2DoF system (for the simulation to run quickly).

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '2dof', ...
                                       'flex_top_type', '3dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', '2dof', ...
                                       'controller_type', 'hac-iff-struts');
```

The reference path as well as the measured motion are compared in Figure 3.9.

The motion errors are computed and shown in Figure 3.10. It is clear that the hexapod is indeed tracking the reference path. However, in this simulation, no disturbances are included nor sensor noises.

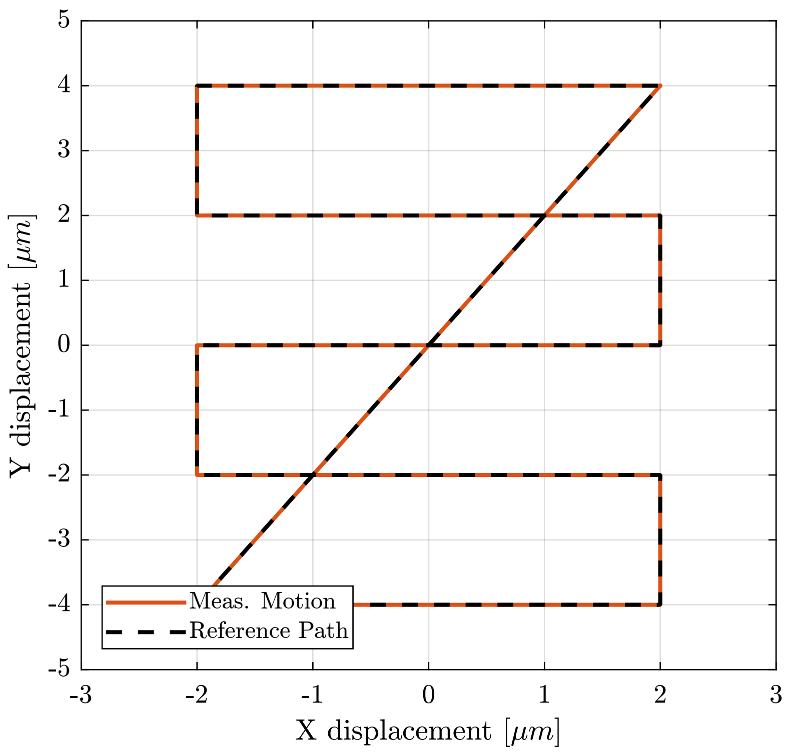


Figure 3.9: Simulated Y-Z motion

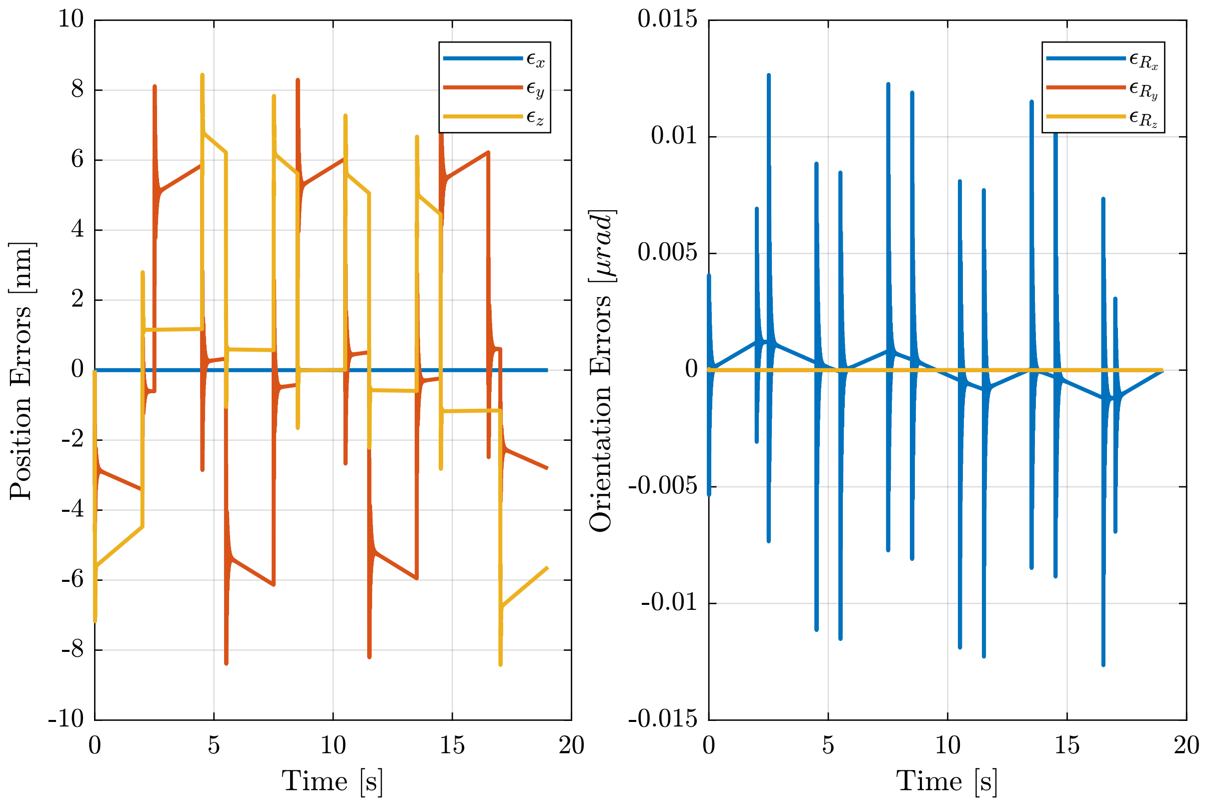


Figure 3.10: Positioning errors as a function of time

3.2.2 Tilt Scans

3.2.3 “NASS” reference path

In this section, a reference path that “draws” the work “NASS” is developed.

First, a series of points representing each letter are defined. Between each letter, a negative Z motion is performed.

Matlab
```
%% List of points that draws "NASS"
ref_path = [ ...
    0, 0,0; % Initial Position
    0,0,1; 0,4,1; 3,0,1; 3,4,1; % N
    3,4,0; 4,0,0; % Transition
    4,0,1; 4,3,1; 5,4,1; 6,4,1; 7,3,1; 7,2,1; 4,2,1; 4,3,1; 5,4,1; 6,4,1; 7,3,1; 7,0,1; % A
    7,0,0; 8,0,0; % Transition
    8,0,1; 11,0,1; 11,2,1; 8,2,1; 8,4,1; 11,4,1; % S
    11,4,0; 12,0,0; % Transition
    12,0,1; 15,0,1; 15,2,1; 12,2,1; 12,4,1; 15,4,1; % S
    15,4,0;
           ];

%% Center the trajectory arround zero
ref_path = ref_path - (max(ref_path) - min(ref_path))/2;

%% Define the X-Y-Z cuboid dimensions containing the trajectory
X_max = 10e-6;
Y_max =  4e-6;
Z_max =  2e-6;

ref_path = ([X_max, Y_max, Z_max]./max(ref_path)).*ref_path; % [m]
```

Then, using the `generateXYZTrajectory` function, the 6×1 trajectory signal is computed.

Matlab
```
%% Generating the trajectory
Rx_nass = generateXYZTrajectory('points', ref_path);
```

The trajectory in the X-Y plane is shown in Figure 3.11 (the transitions between the letters are removed).

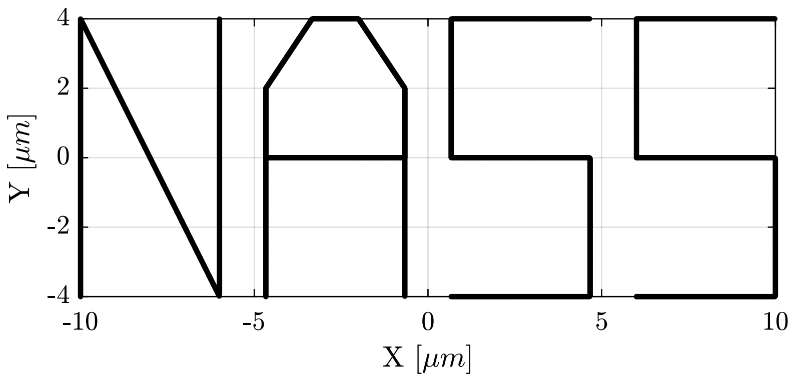


Figure 3.11: Reference path corresponding to the “NASS” acronym

It can also be better viewed in a 3D representation as in Figure 3.12.

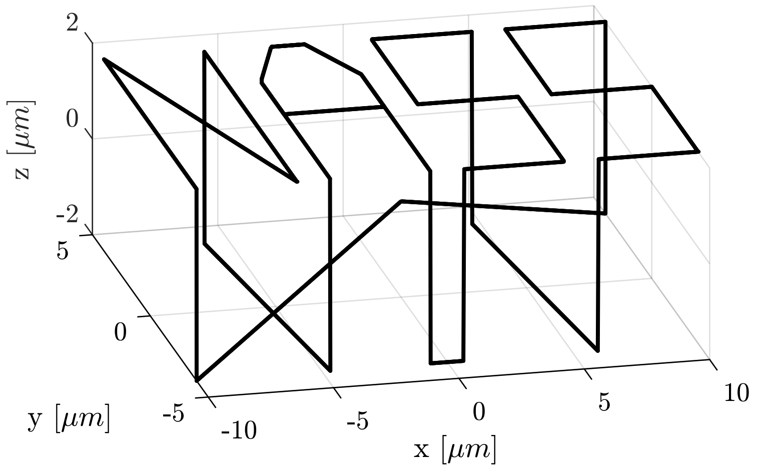


Figure 3.12: Reference path that draws “NASS” - 3D view

3.3 First Experimental Tests with the HAC

Both the Integral Force Feedback controller (developed in Section 2.3) and the high authority controller working in the frame of the struts (developed in Section 3.1) are implemented experimentally.

3.3.1 Initial Controller

The controller designed in Section 3.1 is implemented experimentally and some reference tracking tests are performed.

```matlab
%% Load the experimental data
load('hac_iff_struts_yz_scans.mat', 't', 'de')
```

The position of the top-platform is estimated using the Jacobian matrix:

```matlab
%% Pose of the top platform from the encoder values
load('jacobian.mat', 'J');
Xe = [inv(J)*de']';
```

The reference path as well as the measured position are partially shown in the Y-Z plane in Figure 3.13.

Important

It is clear from Figure 3.13 that the position of the nano-hexapod effectively tracks to reference signal. However, oscillations with amplitudes as large as 50nm can be observe.

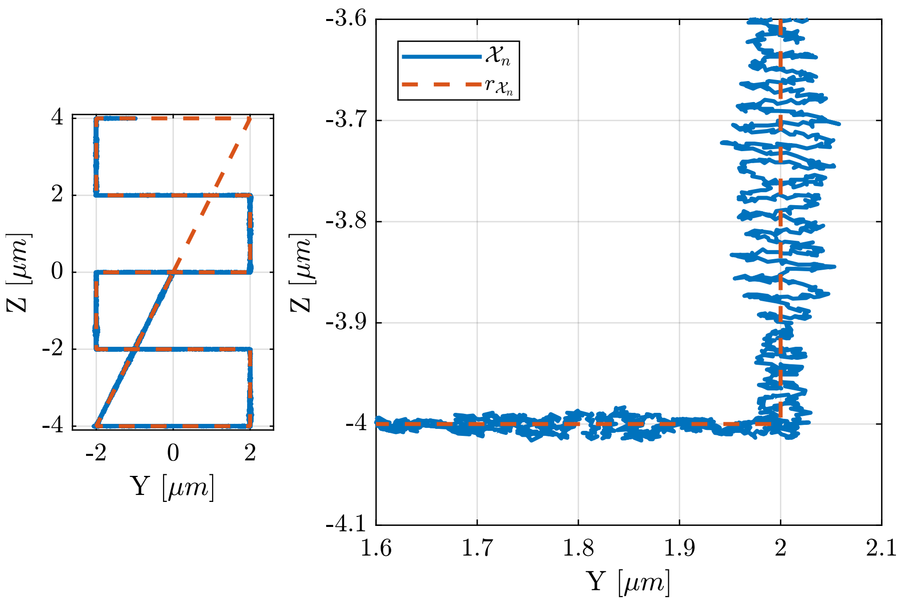


Figure 3.13: Measured position $\boldsymbol{\mathcal{X}}_n$ and reference signal $\boldsymbol{r}_{\mathcal{X}_n}$ in the Y-Z plane - Zoom on a change of direction

It turns out that the frequency of these oscillations is 100Hz which is corresponding to the crossover frequency of the High Authority Control loop. This clearly indicates poor stability margins. In the next section, the controller is re-designed to improve the stability margins.

3.3.2 Controller with increased stability margins

The High Authority Controller is re-designed in order to improve the stability margins.

```
Matlab
%% Lead
a  = 5;  % Amount of phase lead / width of the phase lead / high frequency gain
wc = 2*pi*110; % Frequency with the maximum phase lead [rad/s]

H_lead = (1 + s/(wc/sqrt(a)))/(1 + s/(wc*sqrt(a)));

%% Low Pass Filter
H_lpf = 1/(1 + s/2/pi/300);

%% Notch
gm = 0.02;
xi = 0.5;
wn = 2*pi*700;

H_notch = (s^2 + 2*gm*xi*wn*s + wn^2)/(s^2 + 2*xi*wn*s + wn^2);

%% HAC Controller
Khac_iff_struts = -2.2e4 * ... % Gain
            H_lead * ...       % Lead
            H_lpf * ...        % Lead
            H_notch * ...      % Notch
            (2*pi*100/s) * ... % Integrator
            eye(6);            % 6x6 Diagonal
```

The bode plot of the new loop gain is shown in Figure 3.14.

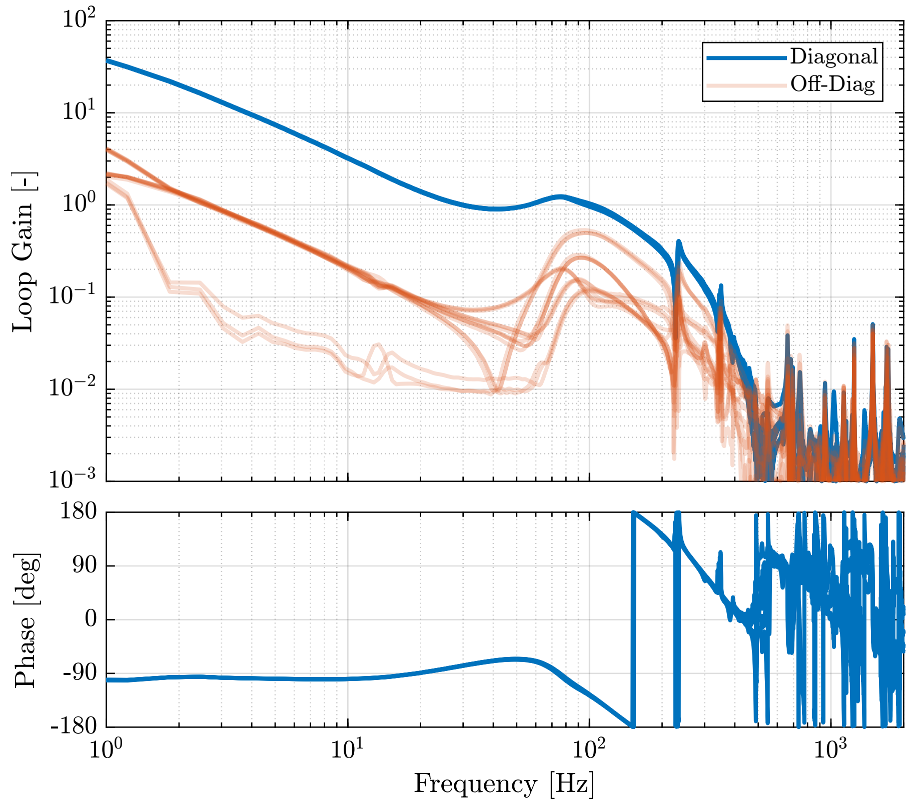


Figure 3.14: Loop Gain for the updated decentralized HAC controller

This new controller is implemented experimentally and several tracking tests are performed.

```
Matlab
%% Load Measurements
load('hac_iff_more_lead_nass_scan.mat', 't', 'de')
```

The pose of the top platform is estimated from the encoder position using the Jacobian matrix.

```
Matlab
%% Compute the pose of the top platform
load('jacobian.mat', 'J');
Xe = [inv(J)*de']';
```

The measured motion as well as the trajectory are shown in Figure 3.15.

The trajectory and measured motion are also shown in the X-Y plane in Figure 3.16.

The orientations errors as a function of time are shown in Figure 3.17.

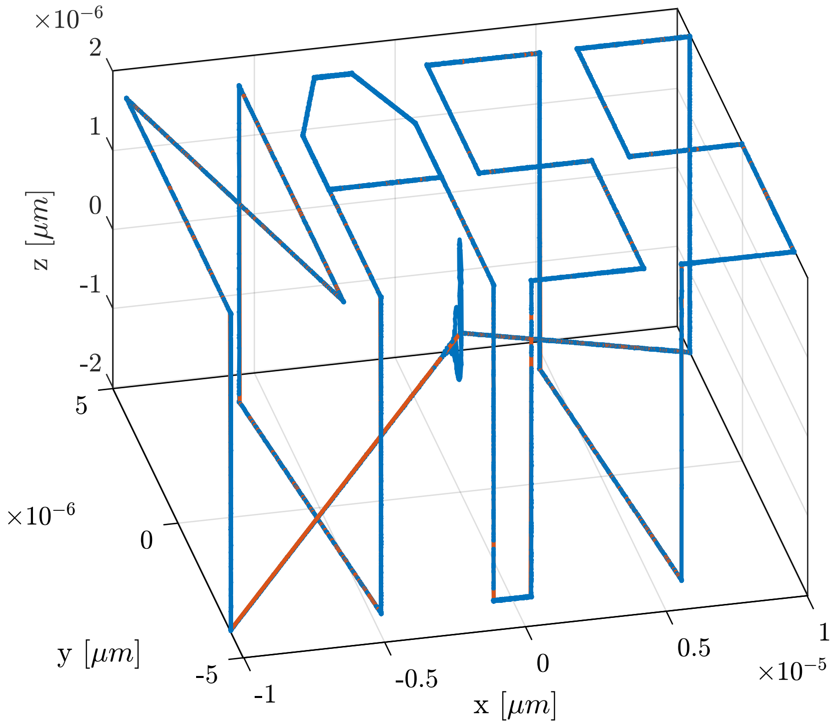


Figure 3.15: Measured position $\boldsymbol{\mathcal{X}}_n$ and reference signal $\boldsymbol{r}_{\mathcal{X}_n}$ for the "NASS" trajectory

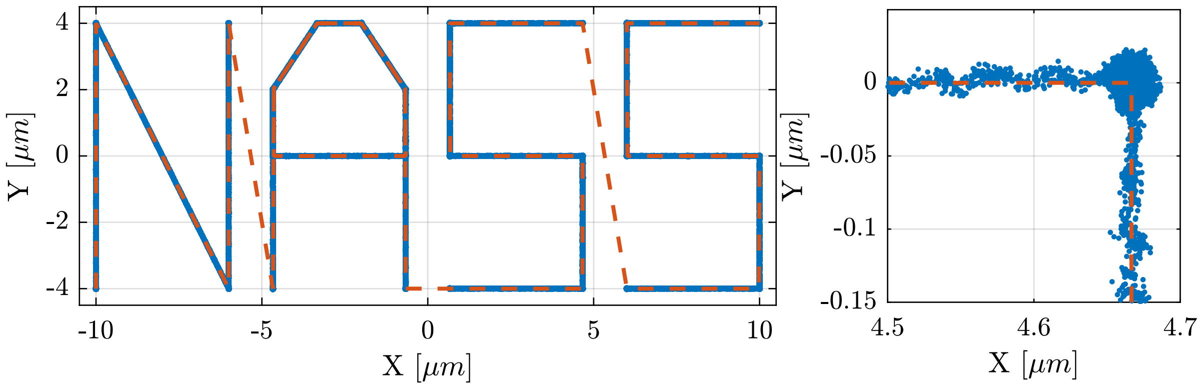


Figure 3.16: Reference path and measured motion in the X-Y plane

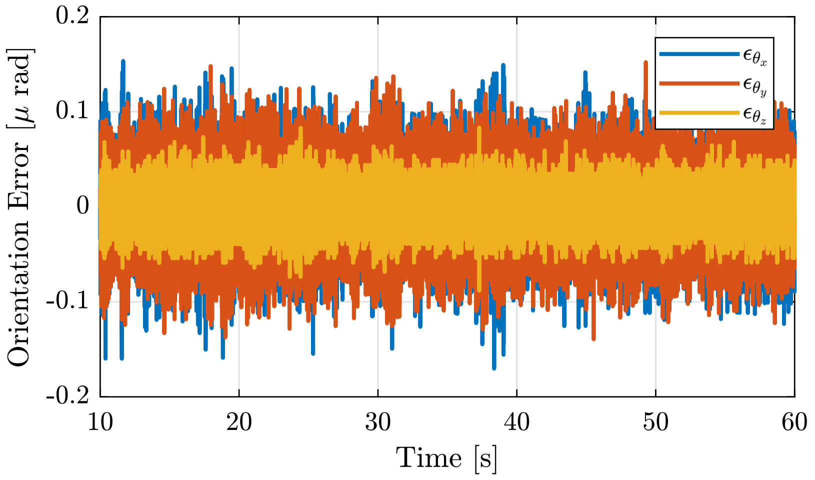


Figure 3.17: Orientation errors as a function of time during the “NASS” trajectory

Important

Using the updated High Authority Controller, the nano-hexapod can follow trajectories with high accuracy (the position errors are in the order of 50nm peak to peak, and the orientation errors 300nrad peak to peak).

4 Functions

4.1 generateXYZTrajectory

Function description

```
Matlab
function [ref] = generateXYZTrajectory(args)
% generateXYZTrajectory -
%
% Syntax: [ref] = generateXYZTrajectory(args)
%
% Inputs:
%    - args
%
% Outputs:
%    - ref - Reference Signal
```

Optional Parameters

```
Matlab
arguments
    args.points double {mustBeNumeric} = zeros(2, 3) % [m]

    args.ti    (1,1) double {mustBeNumeric, mustBePositive} = 1 % Time to go to first point and after last point [s]
    args.tw    (1,1) double {mustBeNumeric, mustBePositive} = 0.5 % Time wait between each point [s]
    args.tm    (1,1) double {mustBeNumeric, mustBePositive} = 1 % Motion time between points [s]

    args.Ts    (1,1) double {mustBeNumeric, mustBePositive} = 1e-3 % Sampling Time [s]
end
```

Initialize Time Vectors

```
Matlab
time_i = 0:args.Ts:args.ti;
time_w = 0:args.Ts:args.tw;
time_m = 0:args.Ts:args.tm;
```

XYZ Trajectory

Matlab
```
% Go to initial position
xyz = (args.points(1,:))'*(time_i/args.ti);

% Wait
xyz = [xyz, xyz(:,end).*ones(size(time_w))];

% Scans
for i = 2:size(args.points, 1)
    % Go to next point
    xyz = [xyz, xyz(:,end) + (args.points(i,:)' - xyz(:,end))*(time_m/args.tm)];
    % Wait a litle bit
    xyz = [xyz, xyz(:,end).*ones(size(time_w))];
end

% End motion
xyz = [xyz, xyz(:,end) - xyz(:,end)*(time_i/args.ti)];
```

Reference Signal

Matlab
```
t = 0:args.Ts:args.Ts*(length(xyz) - 1);
```

Matlab
```
ref = zeros(length(xyz), 7);

ref(:, 1) = t;
ref(:, 2:4) = xyz';
```

4.2 `generateYZScanTrajectory`

Function description

Matlab
```
function [ref] = generateYZScanTrajectory(args)
% generateYZScanTrajectory -
%
% Syntax: [ref] = generateYZScanTrajectory(args)
%
% Inputs:
%    - args
%
% Outputs:
%    - ref - Reference Signal
```

Optional Parameters

Matlab
```
arguments
    args.y_tot (1,1) double {mustBeNumeric} = 10e-6 % [m]
    args.z_tot (1,1) double {mustBeNumeric} = 10e-6 % [m]

    args.n     (1,1) double {mustBeInteger, mustBePositive} = 10 % [-]

    args.Ts    (1,1) double {mustBeNumeric, mustBePositive} = 1e-4 % [s]

    args.ti    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
    args.tw    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
    args.ty    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
    args.tz    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
end
```

Initialize Time Vectors

Matlab
```
time_i = 0:args.Ts:args.ti;
time_w = 0:args.Ts:args.tw;
time_y = 0:args.Ts:args.ty;
time_z = 0:args.Ts:args.tz;
```

Y and Z vectors

Matlab
```
% Go to initial position
y = (time_i/args.ti)*(args.y_tot/2);

% Wait
y = [y, y(end)*ones(size(time_w))];

% Scans
for i = 1:args.n
    if mod(i,2) == 0
        y = [y, -(args.y_tot/2) + (time_y/args.ty)*args.y_tot];
    else
        y = [y,  (args.y_tot/2) - (time_y/args.ty)*args.y_tot];
    end

    if i < args.n
        y = [y, y(end)*ones(size(time_z))];
    end
end

% Wait a litle bit
y = [y, y(end)*ones(size(time_w))];

% End motion
y = [y, y(end) - y(end)*time_i/args.ti];
```

Matlab
```
% Go to initial position
z = (time_i/args.ti)*(args.z_tot/2);

% Wait
z = [z, z(end)*ones(size(time_w))];

% Scans
for i = 1:args.n
    z = [z, z(end)*ones(size(time_y))];

    if i < args.n
        z = [z, z(end) - (time_z/args.tz)*args.z_tot/(args.n-1)];
    end
end

% Wait a litle bit
z = [z, z(end)*ones(size(time_w))];

% End motion
z = [z, z(end) - z(end)*time_i/args.ti];
```

Reference Signal

Matlab
```
t = 0:args.Ts:args.Ts*(length(y) - 1);
```

Matlab
```
ref = zeros(length(y), 7);

ref(:, 1) = t;
ref(:, 3) = y;
ref(:, 4) = z;
```

4.3 `getTransformationMatrixAcc`

Function description

Matlab
```
function [M] = getTransformationMatrixAcc(Opm, Osm)
% getTransformationMatrixAcc -
%
% Syntax: [M] = getTransformationMatrixAcc(Opm, Osm)
%
% Inputs:
%    - Opm - Nx3 (N = number of accelerometer measurements) X,Y,Z position of accelerometers
%    - Opm - Nx3 (N = number of accelerometer measurements) Unit vectors representing the accelerometer orientation
%
% Outputs:
%    - M - Transformation Matrix
```

Transformation matrix from motion of the solid body to accelerometer measurements

Let's try to estimate the x-y-z acceleration of any point of the solid body from the acceleration/angular acceleration of the solid body expressed in $\{O\}$. For any point p_i of the solid body (corresponding to an accelerometer), we can write:

$$\begin{bmatrix} a_{i,x} \\ a_{i,y} \\ a_{i,z} \end{bmatrix} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} + p_i \times \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{4.1}$$

We can write the cross product as a matrix product using the skew-symmetric transformation:

$$\begin{bmatrix} a_{i,x} \\ a_{i,y} \\ a_{i,z} \end{bmatrix} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & p_{i,z} & -p_{i,y} \\ -p_{i,z} & 0 & p_{i,x} \\ p_{i,y} & -p_{i,x} & 0 \end{bmatrix}}_{P_{i,[\times]}} \cdot \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{4.2}$$

If we now want to know the (scalar) acceleration a_i of the point p_i in the direction of the accelerometer direction $\hat{s}_i$, we can just project the 3d acceleration on $\hat{s}_i$:

$$a_i = \hat{s}_i^T \cdot \begin{bmatrix} a_{i,x} \\ a_{i,y} \\ a_{i,z} \end{bmatrix} = \hat{s}_i^T \cdot \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} + \left(\hat{s}_i^T \cdot P_{i,[\times]}\right) \cdot \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{4.3}$$

Which is equivalent as a simple vector multiplication:

$$a_i = \begin{bmatrix} \hat{s}_i^T & \hat{s}_i^T \cdot P_{i,[\times]} \end{bmatrix} \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} \hat{s}_i^T & \hat{s}_i^T \cdot P_{i,[\times]} \end{bmatrix} {}^O\vec{x} \tag{4.4}$$

And finally we can combine the 6 (line) vectors for the 6 accelerometers to write that in a matrix form. We obtain Eq. (4.5).

Important

The transformation from solid body acceleration ${}^O\vec{x}$ from sensor measured acceleration $\vec{a}$ is:

$$\vec{a} = \underbrace{\begin{bmatrix} \hat{s}_1^T & \hat{s}_1^T \cdot P_{1,[\times]} \\ \vdots & \vdots \\ \hat{s}_6^T & \hat{s}_6^T \cdot P_{6,[\times]} \end{bmatrix}}_{M} {}^O\vec{x} \tag{4.5}$$

with $\hat{s}_i$ the unit vector representing the measured direction of the i'th accelerometer expressed in frame $\{O\}$ and $P_{i,[\times]}$ the skew-symmetric matrix representing the cross product of the position of the i'th accelerometer expressed in frame $\{O\}$.

Let's define such matrix using matlab:

Matlab
```matlab
M = zeros(length(Opm), 6);

for i = 1:length(Opm)
    Ri = [0,          Opm(3,i), -Opm(2,i);
         -Opm(3,i),  0,          Opm(1,i);
          Opm(2,i), -Opm(1,i),  0];
    M(i, 1:3) = Osm(:,i)';
    M(i, 4:6) = Osm(:,i)'*Ri;
end
```

Matlab
```
end
```

4.4 `getJacobianNanoHexapod`

Function description

Matlab
```
function [J] = getJacobianNanoHexapod(Hbm)
% getJacobianNanoHexapod -
%
% Syntax: [J] = getJacobianNanoHexapod(Hbm)
%
% Inputs:
%    - Hbm - Height of {B} w.r.t. {M} [m]
%
% Outputs:
%    - J - Jacobian Matrix
```

Transformation matrix from motion of the solid body to accelerometer measurements

Matlab
```
Fa = [[-86.05,  -74.78, 22.49],
      [ 86.05,  -74.78, 22.49],
      [ 107.79, -37.13, 22.49],
      [ 21.74,  111.91, 22.49],
      [-21.74,  111.91, 22.49],
      [-107.79, -37.13, 22.49]]'*1e-3; % Ai w.r.t. {F} [m]

Mb = [[-28.47, -106.25, -22.50],
      [ 28.47, -106.25, -22.50],
      [ 106.25, 28.47,  -22.50],
      [ 77.78,  77.78,  -22.50],
      [-77.78,  77.78,  -22.50],
      [-106.25, 28.47,  -22.50]]'*1e-3; % Bi w.r.t. {M} [m]

H = 95e-3; % Stewart platform height [m]
Fb = Mb + [0; 0; H]; % Bi w.r.t. {F} [m]

si = Fb - Fa;
si = si./vecnorm(si); % Normalize

Bb = Mb - [0; 0; Hbm];

J = [si', cross(Bb, si)'];
```